University of Zurich[UZH]

Department of Informatics

Tensor Methods for High-dimensional Analysis and Visualization

Dissertation submitted to the
Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Rafael Ballester-Ripoll
from Spain

approved in October 2017 at the request of

Prof. Dr. Renato Pajarola
Dr. Peter Lindstrom

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, October 25, 2017

The Chairman of the Doctoral Board: Prof. Dr. Sven Seuken

ABSTRACT

Most visual computing domains are witnessing a steady growth in data set size, complexity, and dimensionality. Flexible and scalable mathematical models that can efficiently compress, process, store, manipulate, retrieve, and visualize such data are therefore of paramount importance, especially for higher dimensions. In this context, tensor decompositions constitute a promising mathematical framework for compactly representing and operating on both dense and sparse data. Initially proposed as an extension of the concept of matrix decomposition for three and more dimensions, they have found various applications in data-intensive visualization, machine learning, and high-dimensional signal processing.

This thesis aims to help bridge these aspects and tackle modern visual computing challenges under the paradigm of a common representation format, namely tensors. Many kinds of data admit a natural representation as higher-order tensors and/or can be parametrized, learned, or interpolated in the form of compact tensor models. Numerous tools that are native and unique to said decompositions exist for analysis and visualization. They can be exploited as soon as the known ground-truth is abstracted into this kind of reduced representation.

To this end we develop a volume compression algorithm tailored to high reduction rates in visualization applications; we explore compressed-domain processing possibilities including multiresolution convolution, derivation, integration, and summed area tables; we produce visualization diagrams directly from compressed tensors via interactive reconstruction; and we propose sensitivity analysis algorithms for model interpretation and knowledge discovery. Emphasis is placed on

compactness and interactivity and is addressed via careful tensor format selection and model fitting, as well as a range of auxiliary technical tools including adaptive quantization, multiresolution data management, parallelized multilinear algebra operations, and others. We conclude that the tensor methods adopted result in a viable and fruitful toolbox for data of diverse origin, size, dimensionality, resolution, and sparsity.

KURZFASSUNG

Datensätze in den meisten Bereichen des Visual Computing zeigen ein stetiges Wachstum in Grösse, Komplexität und Dimensionalität. Flexible und skalierbare mathematische Modelle, die es erlauben diese Daten auf effiziente Weise zu komprimieren, verarbeiten, speichern, manipulieren, abzurufen und zu visualisieren, sind daher von grösster Wichtigkeit, besonders für höherdimensionale Daten. In diesem Zusammenhang stellen Tensor-Dekompositionen ein vielversprechendes mathematisches Framework für das kompakte Repräsentieren und Bearbeiten von sowohl dicht- als auch dünnbesetzten Datenmodelle. Ursprünglich als Erweiterung des Konzepts der Matrix-Dekomposition für drei und mehr Dimensionen vorgeschlagen, fanden sie zahlreiche Anwendungen in den Bereichen datenintensive Visualisierung, machinelles Lernen und höherdimensionale Signalverarbeitung.

Das Ziel dieser Dissertation ist es dazu beizutragen, diese verschiedenen Aspekte zu überbrücken und Herausforderungen im modernen Visual Computing mithilfe eines gemeinsamen Repräsentationsformats, dem Tensor, zu bewältigen. Viele Formen von Daten erlauben eine natürliche Repräsentation als Tensoren höherer Ordnung und/oder können als kompakte Tensor-Modelle parametrisiert, gelernt oder interpoliert werden. Zahlreiche Werkzeuge, massgeschneidert für diese Dekompositionen, existieren für die Analyse und Visualisierung und können benutzt werden, sobald die bekannten Referenzdaten in diese reduzierte Form der Repräsentation abstrahiert wurden.

Dazu entwickeln wir einen Kompressionsalgorithmus für Volumendaten, der

auf die hohen Kompressionsraten in entsprechenden Visualisierungs-Anwendungen zugeschnitten ist; wir erforschen Möglichkeiten zur direkten Bearbeitung komprimierter Daten, einschliesslich Konvolution in Mehrfachauflösung, Ableitung, Integration und Integralbilder; wir erzeugen Visualisierungs-Diagrammen direkt aus komprimierten Tensoren durch interaktive Rekonstruktion; und wir schlagen Algorithmen vor zur Sensitivitätsanalyse für die Modellinterpretation und Wissensentdeckung. Der Schwerpunkt liegt auf kompakter Repräsentation und Interaktivität, was wir erzielen durch die Auswahl entsprechender Tensorformate und Modellanpassung, sowie auch eine Reihe zusätzlicher Hilfswerkzeuge, einschliesslich solcher zur adaptiven Quantisierung, Management mehrfachaufgelöster Daten, Durchführung parallelisierter Operationen aus der multilinearen Algebra und ähnliches. Wir schlussfolgern dass die Anpassung von Tensor-Methoden zu einem nützlichen und vielfältigen Sortiment an Werkzeugen führt, einsetzbar für Daten unterschiedlicher Herkunft, Grösse, Dimensionalität, Auflösung und Dichte.

ACKNOWLEDGMENTS

First of all I want to express my gratitude towards my advisor, Renato Pajarola. The support and freedom he provided were essential for all the ideas developed in this thesis. He was always open and eager about new lines of research, which is especially important in such a flexible and interdisciplinary topic. I wish to thank all present and past VMML members for the great atmosphere they created and continue to create. I thank David and Enrique for their interactions and sustained work on various fruitful tensor-related projects and results, their ability to get involved in very diverse ideas, and their help on good programming practices and creative thinking. I thank Susanne for helping introduce me to the field and sharing her resources and experience at the beginning of my PhD, and Oliver for his insights during these first years. Special thanks go also to my second advisor, Peter Lindstrom, for his inspiration and knowledgeable input on effective compression. I am also grateful towards Ivan Oseledets for giving me the opportunity of being a visitor at his lab, and towards his students, especially Maxim and Evgeny, for interesting conversations on tensor methods from very different angles.

I give special thanks to my parents and sister for their encouragement and positiveness during this period, to my uncle José Ismael for sharing his deep passion for science and his counseling on research life, and to my wife Dorina for her unwavering support and advice at both the academic and personal levels.

Finally, I acknowledge the Swiss National Science Foundation (grant 200021-132521) and the University of Zurich's Forschungskredit (grant FK-16-012), that helped fund my research during the first and last parts of my PhD, respectively.

CONTENTS

NOTATIONS

Acronyms

ACA	Adaptive cross approximation
ALS	Alternating least squares
ANOVA	Analysis of variance
CP	CANDECOMP/PARAFAC
CUR	Column-row matrix factorization
DCT	Discrete cosine transform
DOG	Difference of Gaussians
ETT	Extended tensor train
FT	Fourier transform
HOOI	Higher-order orthogonal iteration
HOPM	Higher-order power method
HOSVD	Higher-order SVD
HT	Hierarchical Tucker
IH	Integral histogram
LHS	Latin hypercube sampling
LOD	Level of detail
PCA	Principal component analysis
PCE	Polynomial chaos expansion
PSNR	Peak signal-to-noise ratio
QOI	Quantity of interest

RLE	Run-length encoding
RMSE	Root-mean-square error
ROI	Region of interest
SA	Sensitivity analysis
SVD	Singular value decomposition
TT	Tensor train
WT	Wavelet transform

Symbols

$\mathbf{u}$	A vector (order-1 tensor)
$\mathbf{U}$	A matrix (order-2 tensor)
$\mathcal{T}$	A tensor of any order
$\mathbf{U}[i,:]$	NumPy notation: i-th row of a matrix
$\mathbf{u}[0{:}k]$	NumPy notation: the first k elements of a vector
$\mathcal{T}[i]$	i-th slice of a tensor train core along the spatial mode
N	Number of tensor dimensions
I, I_n	Tensor sizes, $1 \leq n \leq N$
R, R_n	Tensor ranks
$< \cdot, \cdot >$	Dot product
$\|\| \cdot \|\|$	Frobenius norm
$\circ$	Element-wise (Hadamard) product
$\odot$	Khatri-Rao product
$\otimes$	Kronecker product
$\times_n$	Tensor-times-matrix along the n-th mode
$\mathbf{U} * \mathbf{u}$	Column-wise convolution of a matrix with a vector
$\dagger$	Moore-Penrose pseudoinverse
ϵ	Relative error between two tensors
$\mathbf{T}_{(n)}$	n-th mode unfolding of a tensor $\mathcal{T}$
$\sigma_i^{(n)}$	i-th generalized singular value along the n-th mode
$\widehat{\ }$	Missing item in a list, e.g. $x_1, \ldots, \widehat{x_n}, \ldots, x_N$
$\mathcal{U}$	Uniform distribution
$\mathcal{N}$	Normal distribution

LIST OF FIGURES

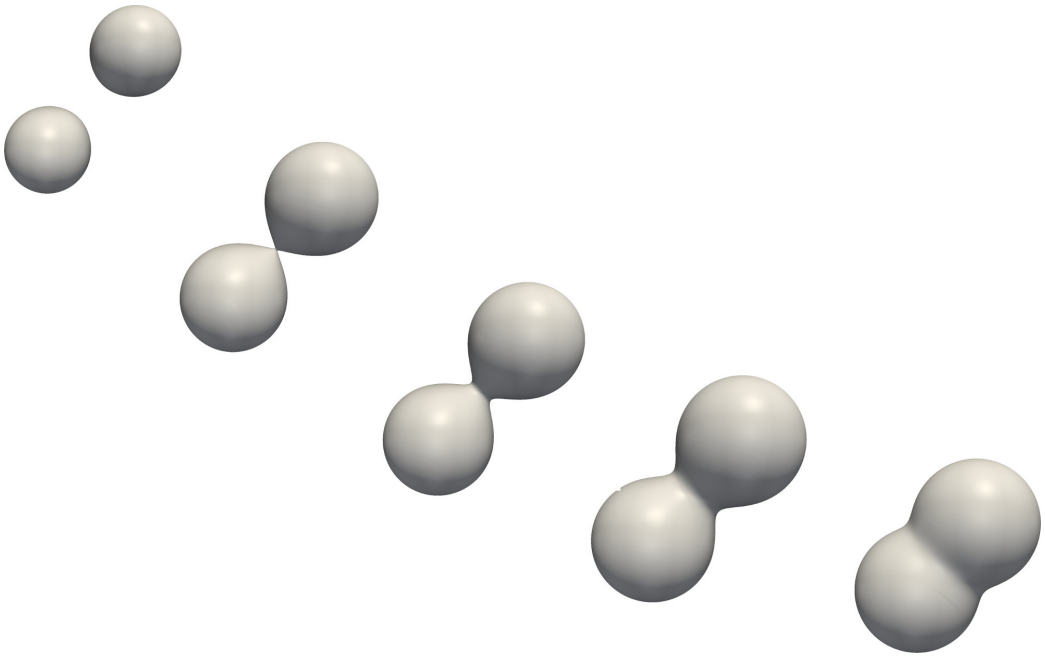
LIST OF TABLES

Part I

Dense Data: Compression and Processing

CHAPTER 1

INTRODUCTION



1.1 High-dimensional Visualization Challenges

Our rapidly increasing ability to generate data is now more than ever unmatched by our ability to make sense out of it. The importance of visual computing has grown in parallel to modern data acquisition capabilities; its applications are far-reaching and have strong ties with computer graphics, machine learning, applied mathematics and statistics, data science, and beyond. A few key aspects of visualization are especially challenging when data is complex and/or high-dimensional:

- Storage and display: memory and network bandwidth bottlenecks are a growing limitation for massive multidimensional data sets, and they increasingly require suitable compression strategies. For example, 3D tomography scans, image stacks, hyperspectral images or time varying data are all much more demanding than their lower-dimensional counterparts.

- Feature extraction: separating important attributes from unimportant background can be a manifold task and can range from noise removal to edge detection to computation of statistical properties.

- Interactivity: real-time exploration continues to be a critical target for many graphics and visualization systems. The well-known *curse of dimensionality* makes it harder for algorithms and architectures to perform satisfactorily with limited computing resources.

- Missing values or regions: the ability to predict or interpolate an output over unobserved points or spaces is a fundamental aim of machine learning and is instrumental in transforming raw sparse data into knowledge.

- Model interpretation: questions such as "*Which independent variables affect more strongly this model's response?*" or "*Are these observations mostly explicable by isolated effects, or are they the result of a more complex interaction network?*" arise frequently during the data exploration process. They move in a virtuous circle with respect to visualization, both benefiting it and arising from it.

1.2 Tensor Decompositions

As a result of the challenges listed above it has become crucial to exploit mathematical tools that can compactly learn, represent and extract features from data. Matrix representations, in particular, are a cornerstone of many applications on bivariate information: signal processing, graphs and networks, dimensionality reduction for unstructured data, recommender systems, etc. High-dimensional data

sets, however, possess a much richer structure than what a matrix can capture. Initially proposed as an extension of the singular value decomposition (SVD) for 3 and more dimensions, *tensor decompositions* and their backbone (*multilinear algebra*) address this issue in a wide range of disciplines. There are several ways to define such extensions, each giving rise to a different approximation model. Pioneering decomposition models such as CANDECOMP/PARAFAC, Tucker or the higher-order singular value decomposition (HOSVD) were originally developed as compact multiway dimensionality reduction techniques. These concepts later spawned many applications in signal processing and data mining. Furthermore, a variety of additional decompositions have been proposed.

More recently, tensor methods have become a versatile and increasingly used tool in computer graphics and visualization. For instance [Wu et al., 2007] and [Wu et al., 2008] developed a hierarchical partitioning for visual data compression, outperforming the wavelet transform (WT) in terms of feature preservation. [Suter et al., 2011a] proposed a graphics processing unit (GPU) decompression algorithm for parallel, real-time large data visualization. [Wetzstein et al., 2012] introduced tensor light field glasses-free 3D displays. [Ruiters and Klein, 2009] and [Ballester-Ripoll and Pajarola, 2016] decomposed bidirectional texture functions, and [Costantini et al., 2008] used HOSVD for fast video texture synthesis. Other applications include denoising [Rajwade et al., 2013], [Zhang et al., 2015a], data recovery and inpainting [Oseledets et al., 2008], [Kressner et al., 2013], [Chen et al., 2014], [Filipović and Jukić, 2015], face transfer [Vlasic et al., 2005], [Vasilescu and Terzopoulos, 2007], deep learning [Novikov et al., 2015], etc. For a more comprehensive compilation of tensor applications in these and other areas we refer the reader to the surveys [de Lathauwer, 2009], [Kolda and Bader, 2009], and [Grasedyck et al., 2013].

In short, tensor representations are compact, lossy for most purposes, and offer multiple ways to manipulate data in its compressed format.

1.3 Research Questions

Despite the past developments on tensor theory and applications in a range of scientific disciplines, two important research questions remain open as we detail next.

RQ1 *Are tensor decompositions applicable for interactive visualization of diverse data types (in terms of sparsity, dimensionality, size, etc.)?* Numerous tensor-based applications have been proposed that work for specific kinds of visual data, but efforts towards one single pipeline for sampling, analysis and visualization are scarce as of yet.

RQ2 *Is there a flexible enough specific tensor format that can work as a standard representation for such distinct visual data types?* Several decompositions exist that are tailored to varying settings in visual computing. But is there a "sweet-spot" model that can best serve as a golden standard?

We answer positively and constructively the first question by defining a visualization pipeline that has tensor decompositions at its core, along with a set of tools and numerical recipes that can deal with different analysis and visualization purposes: compression, statistical properties, global optimization, interactive exploration, signal processing, etc.

The answer to the second question is also positive, but two-fold. On the one hand we favor mainly two models: the *Tucker model* (and its particular case the *CP model*) for lower-dimensional and large-scale dense data sets, and the *TT model* which is more suitable for higher-dimensional data. On the other hand, both models are close relatives within the context of *tensor networks*, and can be seen as particular cases of a certain common format as we will describe later in this thesis.

1.4 Proposed Framework

This dissertation revolves around one central concept, i.e. tensor decomposition for multidimensional visual data, and the unique possibilities that such a flexible and rich mathematical framework brings about. The proposed pipeline (Fig. 1.1) is as follows:

1. As an input we have *multidimensional data*, often complex and large-scale and sometimes sparse, that we need to handle, visualize, and understand;

2. There is a *learning* stage, in which a model is fitted that can generalize the data by removing the irrelevant structures (noise, redundant information) while keeping its essential behavior and interesting features;

3. A common representation, a *tensor decomposition*, compactly captures the aforementioned features under the highly versatile *low tensor rank assumption* (namely, the premise that the data of interest admits a sufficiently good low-rank tensor expansion);

4. Owing to the decomposition's reduced size and unique mathematical properties, efficient *compressed-domain operations* are designed and applied that reveal abstractions and its essential attributes;

5. The features extracted are communicated to a user via modern computer graphics and visualization diagrams and techniques, thereby facilitating responsive knowledge discovery and intuitive understanding of the initial data set.

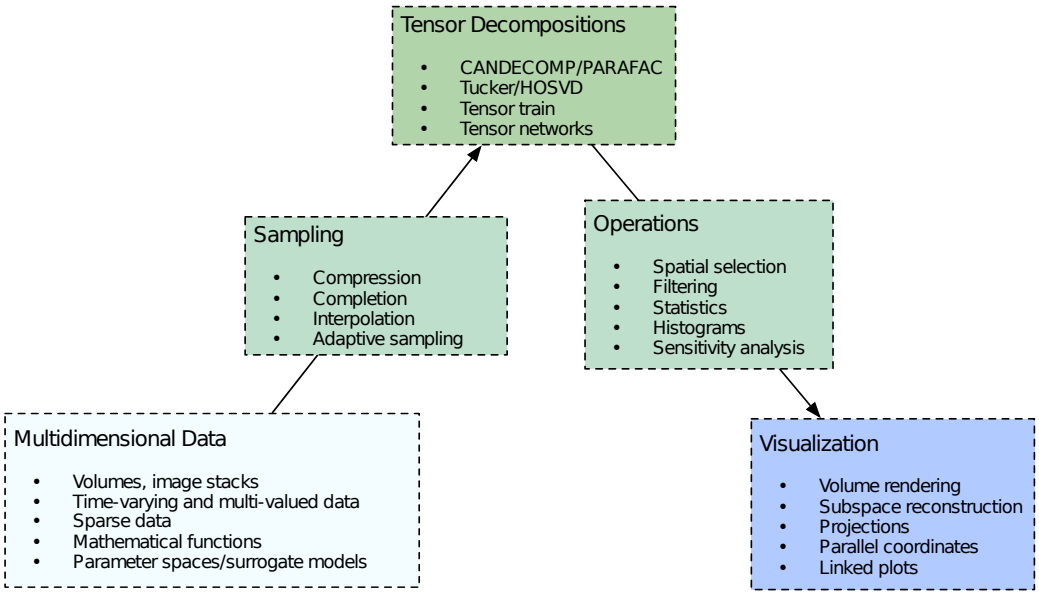


Figure 1.1: *Tensor decompositions are the fundamental link in the proposed high-dimensional visualization pipeline. Several tools exist for fitting tensors to various kinds of data, producing common compressed formats with advantages such as multilinearity, efficient decompression and optimization algorithms.*

1.5 Contributions

The pipeline we just outlined consists of several contribution units. Even though each of them focuses on a different challenge, they share a number of multilinear algebra techniques. Here we list the main contributions proposed in this dissertation, broken down according to their scope, along with their corresponding scientific publications.

Multidimensional Compression

- "*Lossy Volume Compression Using Tucker Truncation and Thresholding*" [Ballester-Ripoll and Pajarola, 2015]

- "TTHRESH*: Tensor Compression for Multidimensional Visual Data*" [Ballester-Ripoll et al., 2017]

We have investigated the benefits of Tucker core thresholding and concluded that it outperforms earlier truncation-based approaches in terms of quality vs. compression ratio. This has led us to introducing TTHRESH (from *Tucker thresholding*), a lossy compression algorithm for N-dimensional data over regular grids. It leverages the Tucker decomposition together with adaptive quantization, run-length and Huffman encoding to store the transform coefficients' relative positions as sorted by their magnitude. The proposed algorithm is the first of its kind that supports arbitrary target accuracy via adaptive quantization. Previous related approaches fixed the number q of quantization bits per transformed coefficient, and sometimes even the number of ranks. Instead, our method improves the compression-accuracy trade-off curve by using thresholding granularity at the single-coefficient level combined with adaptive selection of q.

Our scheme outperforms other state-of-the-art volume compressors at low-to-medium bit rates, as required in data archiving and distribution for e.g. visualization purposes. Further advantages of the compression format include a) extremely fine bit-rate selection granularity; b) upper-bounded resulting L^2 error; and c) support for arbitrary number of dimensions.

Multiresolution Filtering

- "*Multiresolution Volume Filtering in the Tensor Compressed Domain*" [Ballester-Ripoll et al., 2017]

Signal processing and filter operations are important tools for visual data processing and analysis. Due to GPU memory and bandwidth limitations, it is a challenge to apply interactively complex filter operators to large-scale volume data. We propose a novel Tucker-based multiresolution filtering algorithm integrated into an interactive volume visualization system. The raw volume data is decomposed offline into a compact hierarchical (octree-based) tensor approximation model. We then demonstrate how convolution filter operators can effectively be applied in the compressed domain. To prevent aliasing due to multiresolution filtering, our solution filters at the full spatial volume resolution at a very low cost in the compressed domain and reconstructs and displays the filtered result at variable level-of-detail. The proposed system is scalable, allowing interactive display and filtering of large volume datasets that may exceed the available GPU memory.

Compressed Histograms

- "*Tensor Decompositions for Integral Histogram Compression and Look-Up*" [Ballester-Ripoll and Pajarola, 2017]

Histograms are a fundamental tool for multidimensional data analysis and processing, and many applications in graphics and visualization rely on computing histograms over large regions of interest (ROI). Integral histograms (IH) greatly accelerate the calculation in the case of rectangular regions, but come at a large extra storage cost. Based on the Tucker and tensor train decomposition models, we propose a new compression and retrieval algorithm to reduce the overall IH memory usage by several orders of magnitude at a user-defined accuracy. We encode the borders of the desired rectangular ROI in the IH tensor-compressed domain and reconstruct the target histogram at a high speed which is independent of the region size. By tensor-decomposing the ROIs, we furthermore generalize our method to support regions of arbitrary shape, rather than solely rectangles.

Visualizing Tensor-based Surrogate Models

- "*A Surrogate Visualization Model Using the Tensor Train Format*" [Ballester-Ripoll et al., 2016]

Complex simulations and numerical experiments typically rely on a number of parameters and have an associated score function, e.g. with the goal of maximizing accuracy or minimizing computation time. However, the influence of each individual parameter is often poorly understood *a priori* and the joint parameter space can be difficult to explore, visualize and optimize. We model this space as an N-dimensional black-box tensor train (TT) to be learned from a sparse set of samples. Upon learning and compactly expressing this space as a *surrogate visualization model*, informative subspaces are interactively reconstructed and navigated in the form of charts, images, surface plots, etc. By exploiting efficient operations in the TT format, we are able to produce diagrams such as parallel coordinates, bivariate projections and dimensional stacking out of highly-compressed parameter spaces. We demonstrate the proposed framework with several scientific simulations that contain up to 14 parameters and billions of tensor grid points.

Sensitivity Analysis of Tensor Surrogates

- "*Sobol Tensor Trains for Global Sensitivity Analysis*" [Ballester-Ripoll et al., 2017]

Sobol indices are a widespread quantitative measure for variance-based global sensitivity analysis (SA), but computing and utilizing them remains challenging for high-dimensional systems. We propose the TT model as a common interface for Sobol index computation. We introduce the *Sobol tensor train*, which compactly represents a range of variance-based Sobol indices for all possible joint

variable interactions. Our formulation allows efficient aggregation and subselection operations as we are able to obtain related indices (closed and total indices, superset importance measures) at negligible cost. Furthermore, we exploit an existing global optimization algorithm within the TT framework for variable selection and model analysis tasks. Our algorithms are demonstrated in various analytic and real-world engineering simulations and data sets.

1.6 Dissertation Overview

The dissertation is structured in two parts. Both parts follow the proposed pipeline (Fig. 1.1) and are equally concerned with model building, operations in the compressed domain, and interactive visualization.

Part I deals with large-scale compression and manipulation: a spatial data set is available in a full format and is a priori too large to handle, process, or visualize.

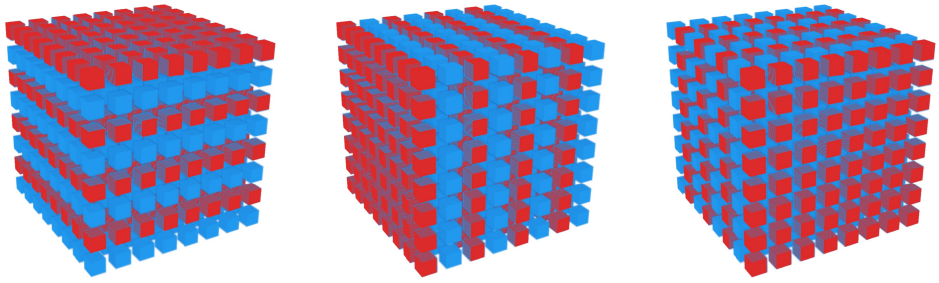
- Ch. 2 introduces the most relevant models for the dissertation: CP, Tucker, TT, and a few others. It also outlines a range of operations that can be applied to compressed tensors.
- Ch. 3 covers the attractive properties that make Tucker-based compression possible and describes the TTHRESH algorithm in detail, including numerical and visual results.
- Ch. 4 integrates Tucker compression into a multiresolution filtering framework in the form of an out-of-core, octree-based volume rendering system.
- Ch. 5 delves further into compressed-domain manipulation in order to exploit integration and summed area table look-up, and derives a retrieval system for approximate histograms.

Part II is concerned with sparse data that must be interpolated directly in a compressed format; i.e. the full data set is never available or even fully reconstructed.

- Ch. 6 exploits tensor learning techniques and adaptive sampling and introduces visualization diagrams that can be efficiently produced from tensors in their compressed form.
- Ch. 7 develops algorithms for sensitivity analysis and model interpretation, based on a novel compressed representation that stores all possible joint interactions between variables.
- Finally, Ch. 8 concludes the dissertation with a summary and gives directions for future work.

CHAPTER 2

TENSOR MODELS



2.1 Overview

Tensor decompositions express N-dimensional arrays as sums of separable tensors, i.e. $\mathcal{T}[\mathbf{x}] \approx \widetilde{\mathcal{T}}[\mathbf{x}] = \sum_r \mathbf{u}_1^{(r)}[x_1] \cdots \mathbf{u}_N^{(r)}[x_N]$. Each element is thus approximated by a multilinear expression, i.e. by sums of products of the compressed model's parameters. However, there are several ways in which these parameters can interact, be stored and multiplied together; most importantly, the vectors $\mathbf{u}_n^{(r)}$ may be shared and reused across several addends. Each such way gives rise to a different decomposition and a different definition of a *tensor rank*, each with distinct properties, asymptotic computational costs, and specific advantages and disadvantages. We generally follow the notation introduced in a few key literature papers including [de Lathauwer et al., 2000b], [Kolda and Bader, 2009], and [Oseledets, 2011].

2.2 CANDECOMP/PARAFAC

The *canonical decomposition* (CANDECOMP/PARAFAC, or CP, also known as *parallel factor* or *Kruskal decomposition*) was the earliest extension of the concept of matrix rank to more than two dimensions [Hitchcock, 1927], [Harshman, 1970]. CP is thereby the most straight-forward generalization of the SVD to $N \geq 3$, as it extends pairs of left and right eigenvectors to N-tuples of vectors. The approximation is built as a linear combination $\widetilde{\mathcal{T}}$ of N-tuples of disjoint 1D basis functions, arranged in columns in the so-called *factor matrices* $\mathbf{U}^{(n)}$ (or simply *factors*), with weight coefficients $\boldsymbol{\lambda}$:

$$\widetilde{\mathcal{T}}[\mathbf{x}] = \widetilde{\mathcal{T}}[x_1, \ldots, x_N] = \sum_{r=0}^{R-1} \boldsymbol{\lambda}[r] \cdot \mathbf{U}^{(1)}[x_1, r] \cdots \mathbf{U}^{(N)}[x_N, r] \tag{2.1}$$

where R is the *CP rank*, or sometimes just *tensor rank*. This can be written more succinctly with the Kronecker product ($\otimes$) as

$$\widetilde{\mathcal{T}} = \sum_{r=0}^{R-1} \boldsymbol{\lambda}[r] \cdot \mathbf{U}^{(1)}[:, r] \otimes ... \otimes \mathbf{U}^{(N)}[:, r], \tag{2.2}$$

and is illustrated in Fig. 2.1. We can also write Eq. 2.2 using double bracket notation as $\mathcal{T} \approx [[\boldsymbol{\lambda}; \mathbf{U}^{(1)}, ..., \mathbf{U}^{(N)}]]$. The $\boldsymbol{\lambda}$ can be optionally absorbed column-wise by the factors $\mathbf{U}^{(n)}$ and omitted in the notation. For compactness we also use sometimes $\lambda_r := \boldsymbol{\lambda}[r]$ and $\mathbf{U}_r^{(n)} := \mathbf{U}^{(n)}[:, r]$.

One important difference with respect to the SVD is the loss of orthonormality, i.e. $\mathbf{U}^{(n)^T} \cdot \mathbf{U}^{(n)}$ need not be the identity matrix $\mathbf{I}$ for any n-th factor in a CP

decomposition. Also, the set of N-dimensional tensors of fixed CP rank R is not closed in $\mathbb{R}^N$, and thus finding the best rank-R approximation of a given tensor is an ill-posed problem [de Silva and Lim, 2008]. On the positive side, the CP format needs $O(INR)$ elements for storage, i.e. is linear w.r.t. the number of dimensions.

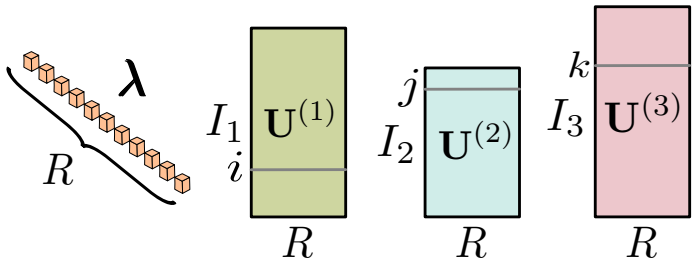


Figure 2.1: *The CP approximation model in 3D, highlighting as gray rows the 3 subspaces that are needed to reconstruct a single voxel* $[i, j, k]$*: the rows* $\mathbf{U}^{(1)}[i, :]$, $\mathbf{U}^{(2)}[j, :]$ *and* $\mathbf{U}^{(3)}[k, :]$*, one from each factor matrix.*

2.3 Tucker

The *Tucker decomposition* [Tucker, 1966] is also known as the *N-mode principal component analysis* (N-PCA), *N-dimensional SVD* (N-SVD), and *low multilinear rank approximation* (LMLRA). Tucker and the closely-related higher-order singular value decomposition (HOSVD [de Lathauwer et al., 2000b]) are among the most popular tensor models of choice in the computer graphics and visualization literature. Tucker generalizes CP by including all interactions between its factor columns, weighted by an N-dimensional core $\mathcal{B}$ of size $R_1 \times ... \times R_N$ (the *Tucker ranks*):

$$\widetilde{\mathcal{T}}[\mathbf{x}] = \sum_{r_1=0}^{R_1-1} \cdots \sum_{r_N=0}^{R_N-1} \mathcal{B}[r_1, \ldots, r_N] \cdot \mathbf{U}^{(1)}[x_1, r_1] \cdot ... \cdot \mathbf{U}^{(N)}[x_N, r_N] \tag{2.3}$$

or, with Kronecker products,

$$\widetilde{\mathcal{T}} = \sum_{r_1=0}^{R_1-1} \cdots \sum_{r_N=0}^{R_N-1} \mathcal{B}[r_1, \ldots, r_N] \cdot \mathbf{U}^{(1)}[:, r_1] \otimes \cdots \otimes \mathbf{U}^{(N)}[:, r_N] \tag{2.4}$$

This format can also be understood via the concept of *multiway projection*, also known as *tensor-times-matrix* product (TTM), which extends the 2D matrix-matrix product. The n-th mode TTM between a tensor $\mathcal{B}$ of size $I_1 \times ... \times I_N$

and a matrix $\mathbf{U}$ of size $J \times I_n$ is a form of *tensor contraction*, usually denoted as $\mathcal{B} \times_n \mathbf{U}$ [de Lathauwer et al., 2000b]. It contracts $\mathbf{U}$ row-wise with $\mathcal{T}$ along its n-th dimension to yield a tensor of size $I_1 \times ...I_{n-1} \times J \times I_{n+1} \times ... \times I_N$. Note that "contraction" here does not refer to size: a TTM operation on a tensor can reduce (when $J < I_n$) as well as increase (when $J > I_n$) its size.

The notation presented allows us to write the Tucker decomposition as a sequence of TTM operations:

$$\widetilde{\mathcal{T}} = \mathcal{B} \times_1 \mathbf{U}^{(1)} \times_2 ... \times_N \mathbf{U}^{(N)} \tag{2.5}$$

or even more compactly, $\widetilde{\mathcal{T}} = [[\mathcal{B}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]]$. See also Fig. 2.2.

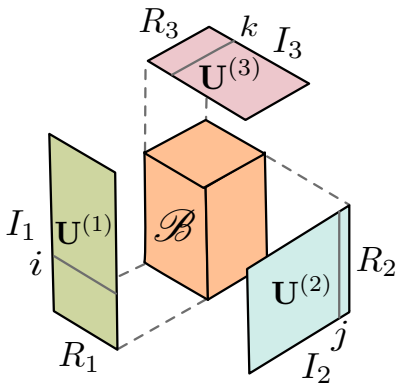


Figure 2.2: *The 3D Tucker approximation model, highlighting the 3 factor rows that are needed to reconstruct a single voxel (i, j, k): the rows $\mathbf{U}^{(1)}[i, :]$, $\mathbf{U}^{(2)}[j, :]$ and $\mathbf{U}^{(3)}[k, :]$ of each factor matrix.*

The Tucker factor matrices can be interpreted as dictionaries: each N-tuple of basis functions yields, via the Kronecker product, a separable N-D array. The set of all possible N-plets (atoms) defines a dictionary (known as *Kronecker dictionary*), and the Tucker core contains the linear coefficients that accompany them.

These equations do not have a unique solution in terms of factors $\mathbf{U}^{(n)}$. In the literature, the Tucker model usually refers to expressing the data in that decomposed form with any factors. However, column-wise orthonormal factors are often sought because they define an immediate one-to-one mapping between any input tensor and its transform. HOSVD is an algorithm for obtaining a particular Tucker decomposition by taking the singular vectors of the unfolded tensor as basis elements. It brings in some benefits, including orthonormal factor matrices and core sparsity (many elements close to zero), and is in many ways the most natural multiway generalization of the matrix SVD [de Lathauwer et al., 2000a]. The HOSVD truncation (i.e. discarding high-rank terms of the HOSVD) is quasi-optimal, in the sense that the resulting error is at most $\sqrt{N}$ times the error of the best possible

Tucker solution with the same rank (see [Hackbusch, 2012], Theorem 10.3). On top of this, the popular higher-order orthogonal iteration (HOOI) [de Lathauwer et al., 2000b] applies HOSVD iteratively to further refine the solution (see also App. A).

The main disadvantage of the Tucker format is that it must store $O(R^N + INR)$ elements, hence it still suffers from the curse of dimensionality w.r.t. to N. In practice it is mostly used for up to a handful of dimensions only, and in this cases it often achieves very attractive compression rates. There are connections between these models and other tools such as the SVD, the Fourier, cosine and wavelet transforms, and polynomial chaos expansions (see also [Ballester-Ripoll et al., 2015], and Sec. 6).

2.4 Tensor Train

As mentioned earlier, classical tensor models have a number of drawbacks regarding algorithmic complexity. No stable procedure exists for decomposing a tensor into CP, and determining the exact number of CP ranks is known to be an ill-posed problem. Conversely, stable algorithms that obtain the Tucker decomposition do exist [de Lathauwer et al., 2000b], but the format still requires exponential space w.r.t. the number of dimensions N. To overcome these issues and unite the advantages of both CP and Tucker, a newer decomposition model named *tensor train* (TT) was recently proposed [Oseledets, 2011]. Previously it was known only in the quantum physics community as *matrix product states* or *linear tensor network*. In the TT model each dimension 1 to N is encoded with one 3D tensor core $\mathcal{T}^{(1)}$ to $\mathcal{T}^{(N)}$. Element-wise,

$$\widetilde{\mathcal{T}}[\mathbf{x}] = \sum_{\mathbf{r}} \mathcal{T}^{(1)}[0, x_1, r_1] \dots \mathcal{T}^{(n)}[r_{n-1}, x_n, r_n] \dots \mathcal{T}^{(N)}[r_{N-1}, x_N, 0] \tag{2.6}$$

where $1 \leq x_n \leq I_n$ for $n = 1, ..., N$. In matrix product notation,

$$\widetilde{\mathcal{T}}[\mathbf{x}] = \mathcal{T}^{(1)}[x_1] \cdot ... \cdot \mathcal{T}^{(N)}[x_N] \tag{2.7}$$

where $\mathcal{T}^{(n)}[x_n]$ is a shorthand for the x_n-th slice along mode 2, i.e. $\mathcal{T}^{(n)}[:, x_n, :]$. See Fig. 2.3 for an illustration. The core dimensions are $R_{n-1} \times I_n \times R_n$ for $n = 1, ..., N$. The integers $R_0, \dots, R_N$ are the *TT ranks*, where $R_0 := R_N := 1$ by convention. More compactly we can also write $\widetilde{\mathcal{T}} = [[\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(N)}]]$. One may equivalently write 2.6 in terms of rank-1 or vectorized tensors; a comprehensive list is available in [Lee and Cichocki, 2017].

TT's main advantage w.r.t. CP is its robustness: the set of tensors with fixed TT rank forms a manifold in $\mathbb{R}^N$, and there exists an error-bounded decomposition algorithm (TT-SVD [Oseledets, 2011], see also App. A). In addition, the

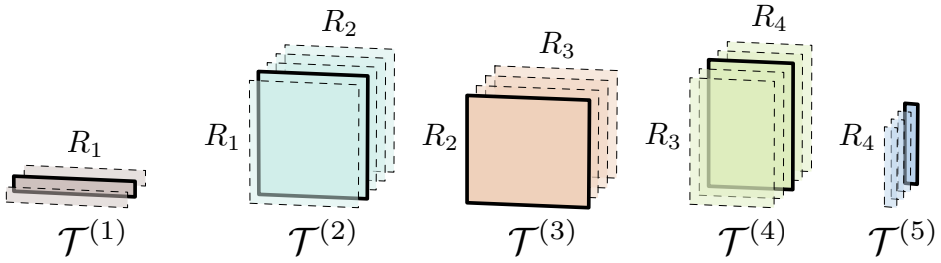


Figure 2.3: *A 5D tensor train of size* $3 \times 5 \times 4 \times 5 \times 4$*. By multiplying the highlighted slices together we obtain the element* $\mathcal{T}[1, 1, 0, 2, 3]$.

TT format needs $O(INR^2)$ elements and thus grows linearly w.r.t the number of dimensions N, unlike Tucker. Note also the speed of TT when reconstructing a single element: only certain slices of the compressed data participate in the calculation, which is a sequence of matrix-vector products requiring $O(NR^2)$ operations. On the other hand, with Tucker the whole core with R^N elements must be traversed. Depending on the data, however, Tucker may need a smaller R than TT for a comparable compression quality.

Since its inception, the decomposition has found use in many different high-dimensional problems, ranging from quantum chemistry [Kazeev et al., 2014] to stochastic partial differential equations [Konakli and Sudret, 2015] or probabilistic graphical models [Novikov et al., 2014], with the latter having applications in image processing and computer vision, among others. A further link exists that connects the tensor train (more specifically its *tensorized* version QTT) with the wavelet transform [Oseledets and Tyrtyshnikov, 2011; Kazeev and Oseledets, 2013].

Several toolboxes and numerous algorithms have been implemented for the CP, Tucker and TT format, see also App. B.

2.5 General Tensor Networks

Both Tucker and TT are particular cases of tensor networks (TN). A TN graph-like representation that defines compactly the structure of a tensor decomposition. Multiarrays are represented by nodes, and physical or virtual dimensions by edges. Physical edges are also called *spatial* or *free* edges. Connecting two edges (free or otherwise) is equivalent to contracting their nodes along a dimension, i.e. it corresponds to canceling two variables in the Einstein summation convention. See Fig 2.4 for several examples.

Other networks include *hierarchical Tucker* (HT for short, also known as *tree tensor network*), *projected entangled pair states* (PEPS), and the *tensor chain*

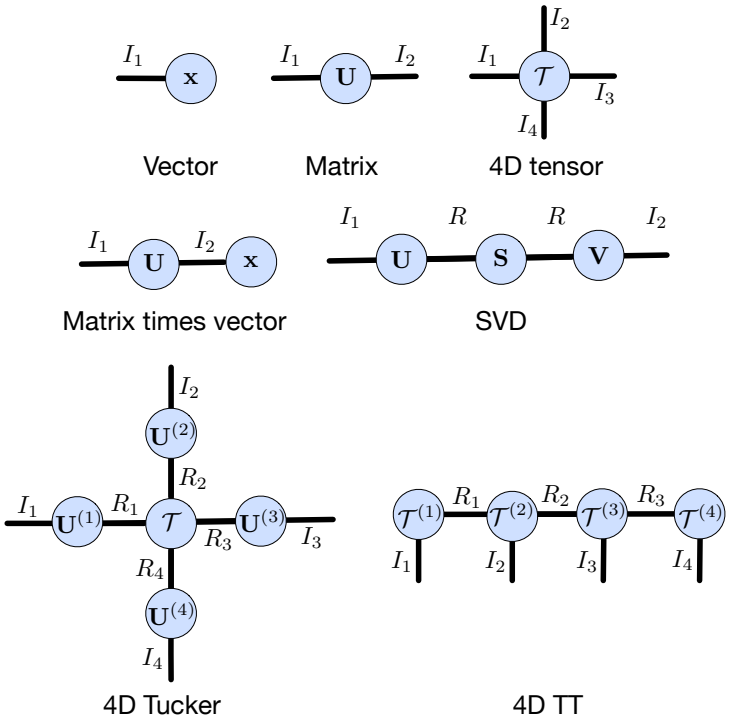


Figure 2.4: *Graphical representation of several tensor networks, ranging from multidimensional objects (top row) to matrix expressions (mid row) to tensor decompositions (bottom row).*

(also known as *tensor ring* [Zhao et al., 2016]). We refer the reader to the survey [Cichocki et al., 2016] and the dissertation [Handschuh, 2015] for more in-depth analyses of tensor networks and their graphical representation, flexibility and growing potential for efficient big data manipulation and optimization.

2.6 Variants and Hybrid Models

Many structural variations and additional constraints can be imposed on these main models to obtain more custom decompositions. For example, one may enforce a block-like structure (block tensor decomposition, or BTD [De Lathauwer, 2008a], [De Lathauwer, 2008b]) or hierarchical/multiresolution partitioning schemes, e.g. [Wu et al., 2008] and TAMRESH [Suter et al., 2013]). Sparsity was imposed by [Ruiters and Klein, 2009] to form a sparse tensor decomposition (STD), which is an independently proposed sparse version of the tensor train. K-clustered tensor approximation (K-CTA) was contributed by [Tsai and Shih, 2012]. It is a semi-sparse decomposition that represents separate data slices via

disjoint slice-wise clusters within the Tucker core. The multiway K-clustered tensor approximation (MK-CTA) [Tsai, 2015] extended this idea to slices along all possible tensor modes. Many of these models emphasize high data reduction rates and especially fast random access (as demanded by interactive surface rendering of 3D scenes).

2.7 Operating with Tensors

2.7.1 Factor Matrix Operations

Certain transformations on the CP/Tucker factor matrices have a useful effect on the output tensor $\widetilde{\mathcal{T}}$ that results from Eqs. 2.1 to 2.5. Factor rows map to axis-aligned hyperslices from the input, while factor columns (ranks) map to axis-aligned hyperslices of the transform. Removing rows prior to reconstruction results in a subtensor of the original input, while removing ranks results in a coarser approximation. Furthermore, if instead of a matrix $\mathbf{U}^{(n)}$ we use a linear combination of its rows $\mathbf{u}^{(n)} := \mathbf{c}^T\mathbf{U}^{(n)}$ (where $\mathbf{c}$ is a vector with I_n entries), then the reconstruction produces the combination of the corresponding hyperslices along the n-th mode. In the Tucker case, this means that $\mathcal{B} \times_1 \mathbf{U}^{(1)} \times_2 ... \times_n \mathbf{u}^{(n)} \times_{n+1} ... \times_N \mathbf{U}^{(N)}$ equals

$$\sum_{k=1}^{I_n} c[k] \cdot \widetilde{\mathcal{T}}[..., k, ...] = \widetilde{\mathcal{T}} \times_n \mathbf{c}^T, \tag{2.8}$$

where we have written a tensor-times-vector (TTV) product on the right-hand side (a mode-n TTV computes the dot product between a vector and each mode-n tensor fiber). For matrices, it holds that $\mathcal{B} \times_1 \mathbf{U}^{(1)} \times_2 ... \times_n (\mathbf{A} \cdot \mathbf{U}^{(n)}) \times_{n+1} ... \times_N \mathbf{U}^{(N)} = \widetilde{\mathcal{T}} \times_n \mathbf{A}$. In particular, we can downsample the factors column-wise to obtain downsampled versions of our data. This principle was first exploited in volume visualization by [Suter et al., 2013] and further analyzed in [Ballester-Ripoll et al., 2015], [Ballester-Ripoll and Pajarola, 2015]. We frequently use similar manipulations in this thesis, especially in Ch. 4 and Ch. 5. Last, note that equivalent spatial manipulation properties hold for many other tensor decompositions; for example in TT one must use combinations of core slices instead of matrix rows.

2.7.2 TT Format

Multiplication/division of a TT tensor by a scalar α is achieved by simply multiplying/dividing one of its cores (say, the first) by α [Oseledets, 2011]. Tensor-tensor addition is written as $(\mathcal{T}_1 + \mathcal{T}_2)[\mathbf{x}] := \mathcal{T}_1[\mathbf{x}] + \mathcal{T}_2[\mathbf{x}]$ and has the following cores:

$$\begin{cases} \left(\begin{array}{c|c} \mathcal{T}_1^{(1)}[x_1] & \mathcal{T}_2^{(1)}[x_1] \end{array} \right) & \text{(first core)} \\ \left(\begin{array}{c|c} \mathcal{T}_1^{(n)}[x_n] & 0 \\ \hline 0 & \mathcal{T}_2^{(n)}[x_n] \end{array} \right) & (1 < n < N) \\ \left(\begin{array}{c} \mathcal{T}_1^{(N)}[x_N] \\ \hline \mathcal{T}_2^{(N)}[x_N] \end{array} \right) & \text{(last core)} \end{cases}$$

To subtract two tensors, we use the same addition method but flip first the second tensor's sign (done by flipping the sign of, say, its first core).

The element-wise (or Hadamard) product $(\mathcal{T}_1 \circ \mathcal{T}_2)[\mathbf{x}] := \mathcal{T}_1[\mathbf{x}] \cdot \mathcal{T}_2[\mathbf{x}]$ arises from a slice-wise Kronecker product:

$$(\mathcal{T}_1^{(1)}[x_1] \otimes \mathcal{T}_2^{(1)}[x_1]) \cdot ... \cdot (\mathcal{T}_1^{(N)}[x_N] \otimes \mathcal{T}_2^{(N)}[x_N]) \tag{2.9}$$

This product has many applications. Convolution between two N-dimensional TT tensors, for example, can be obtained by computing each tensor's Fourier transform (FT) along the spatial dimension on every TT core, followed by element-wise product and inverse FT [Rakhuba and Oseledets, 2015].

2.7.3 Adaptive Cross-Approximation

The so-called *adaptive cross-approximation* (ACA) for the TT format [Oseledets and Tyrtyshnikov, 2010b], [Savostyanov and Oseledets, 2011] is a type of progressive sampling scheme that allows to approximate all mentioned operations and others at cost $O(INR^3)$ at most, i.e. devoid of the curse of dimensionality. These include arbitrary element-wise functions, differentiation, integration, convolution, and more [Cichocki et al., 2016], [Lee and Cichocki, 2017]. The ranks needed may grow as a result of such operations. It is crucial to keep them reasonably low at all stages of any computational pipeline, otherwise the benefits of tensor compression vanish. An error-bounded rounding algorithm called TT-round [Oseledets, 2011] exists to re-compress down any tensor when needed by means of a sequence of $N-1$ SVD truncations. ACA is exploited and covered in more detail in Ch. 6; see also App. A.

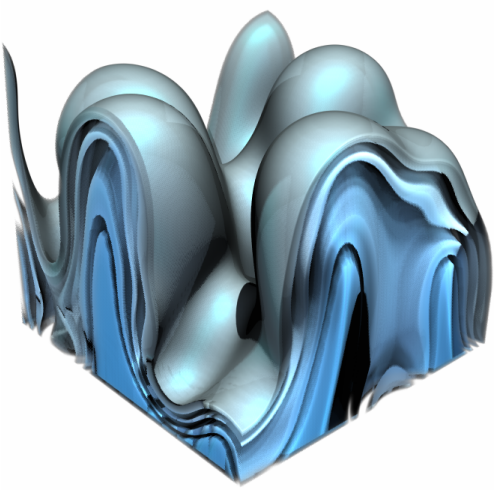
Global Optimization

ACA has been successfully used to find the (approximately) maximal element *in modulus* of a tensor [Dolgov et al., 2014], [Oferkin et al., 2015], as it was empirically found that the subtensors accessed during ACA very often contain such maximal elements. The variant known as rectangular maxvol is a tool even

more efficient for this task [Mikhalev and Oseledets, 2015] and is the one we use (released in [ttp,]). This effectively allows solving global optimization problems in the TT format, a very attractive feature we make use of in Ch. 6 and 7.

CHAPTER 3

TENSOR COMPRESSION



3.1 Overview

As outlined earlier, this part of the dissertation is devoted to dense tensors. The challenge we tackle first is multidimensional compression, which is a significant actor in data-intensive visualization. Two frequent goals in scientific or visual computing applications are a) to reduce the complex initial input to alleviate computational bottlenecks, while b) aiming for a faithful and efficient reconstruction. This is the case when memory or time restrictions are an issue. Lossy compression is often the prescribed strategy, since complex and large data sets rarely benefit much from lossless compression (especially if they use floating-point precision). If the compressed data set is to be used for subsequent computational analysis and/or to be fed into simulation routines, typically very small errors are tolerated. Conversely, if visualization is to follow decompression, then higher error rates are acceptable.

Depending on the specific application, sometimes certain additional targets are desirable. These include fast support for random-access decompression, fine compression rate granularity, asymmetry (faster decompression than compression), bounded error, support for higher-dimensions, ease of parallelization, etc. For these reasons the compression problem is both broad and challenging, and no catch-all solution exists as of yet.

Contribution

We introduce TTHRESH, a tensor compressor for visualization applications whose foremost priority is data reduction at high compression ratios. In particular, while the ratios we achieve at low error tolerance are reasonable, we outperform state-of-the-art methods on the higher error spectrum. Our algorithm also possesses advantages that are inherent to multilinear transforms in general and tensor decompositions in particular, including support for linear manipulation of the data set in the compressed domain (recall Sec. 2.7.1).

Definitions

The RMSE (root-mean-square error) between a tensor of size $I_1 \times \cdots \times I_N$ and an approximation $\widetilde{\mathcal{T}}$ is $\|\mathcal{T} - \widetilde{\mathcal{T}}\|/\sqrt{I_1 \cdots I_N}$. For our experiments we use the normalized PSNR (peak signal-to-noise ratio) in terms of the RMSE as follows:

$$\mathrm{PSNR}(\mathcal{T}, \widetilde{\mathcal{T}}) = 20 \cdot \log_{10} \left(\frac{\max\{\mathcal{T}\} - \min\{\mathcal{T}\}}{2 \cdot \mathrm{RMSE}(\mathcal{T}, \widetilde{\mathcal{T}})} \right) \tag{3.1}$$

3.2 Background

Several 3D lossy compression algorithms have been proposed in the recent literature. ISABELA [Lakshminarasimhan et al., 2011] focuses on spatio-temporal data with heavy high-frequency components; it works by sorting elements into a monotonic curve which is then fitted using B-splines. A more recent example of strategy based on linearization is SZ [Di and Cappello, 2016]. With SZ, each coefficient is either predicted using low-degree polynomials on its preceding values, or truncated in its IEEE 754 binary representation. Sparse coding approaches such as COVRA [Gobbetti et al., 2012] require slow heuristics or greedy algorithms at compression, but are fast to decompress. This makes them suitable for compression-domain direct volume rendering (see also the survey [Balsa Rodríguez et al., 2013]); in particular, COVRA is defined as an octree multiresolution hierarchy.

A prominent and long-standing family of methods are the ones that exploit linear transforms, including very well-known decompositions such as DCT/Fourier or multidimensional wavelets. They are based on the fact that real-world signals tend to be sparse in certain transformed domains. VAPOR [Clyne et al., 2007], for example, uses Haar wavelet-based compression integrated into an interactive volume and flow exploration tool. ZFP [Lindstrom, 2014] is a floating-point compressor using custom (but fixed) transform matrices that emphasizes fast random access and can act as a transparent layer on top of raw C/C++ arrays. It does so via fixed-rate encoding, although a variable-rate variant was also proposed.

More recently, several transform-based compression algorithms have been defined that use data-dependent basis. This is precisely the idea behind the Tucker model. It trades off better transform-domain sparsity for the cost of storing learned bases, which tends to be small (especially for higher dimensions). Early Tucker-based compression approaches for visual spatial data include [Wang and Ahuja, 2004], [Wu et al., 2007] and [Wu et al., 2008]. Progressive tensor rank reduction has been shown to reveal features and structural details at different scales also in volume data [Suter et al., 2010a; Suter et al., 2010b]. Further recent efforts in the context of tensor compression include [Tsai, 2009; Tsai and Shih, 2012; Ballester-Ripoll et al., 2015; Ballester-Ripoll and Pajarola, 2015; Tsai, 2015], [Suter et al., 2011b; Suter et al., 2013] and [Balsa Rodríguez et al., 2014] for interactive volume visualization, and [Wetzstein et al., 2012] for 3D displays.

3.3 Compression with Tucker

3.3.1 Core Reduction

The Tucker decomposition (recall Sec. 2.3) of a tensor $\mathcal{T}$ of size $\mathbf{I} = (I_1, \ldots, I_N)$ is always exact when all ranks are kept:

$$\mathcal{T}[\mathbf{x}] = \sum_{\mathbf{r}=0}^{\mathbf{I}-\mathbf{1}} \mathcal{B}[\mathbf{r}] \cdot \mathbf{U}^{(1)}[x_1, r_1] \cdots \mathbf{U}^{(N)}[x_N, r_N] \tag{3.2}$$

provided that every $\mathbf{U}^{(n)}$ is an invertible matrix of size $I_n \times I_n$ and $\mathcal{B}$ has the same size as $\mathcal{T}$. The factors then define an invertible transformation between $\mathcal{T}$ and $\mathcal{B}$. The HOSVD produces orthogonal factors by taking each $\mathbf{U}^{(n)}$ as the leading left singular vectors of the n-th mode unfolding $\mathbf{T}_{(n)}$ (see also App. A). In other words, it uses the PCA transform matrix of the set of all fibers from $\mathcal{T}$, taken along the n-th mode. One advantage of this tensor formulation is the simplicity of higher-order extensions: the format is flexible and readily applicable to any shape and dimensionality, and the decomposition always exists.

The HOSVD transform coefficients are generally quasi-sparse. The transform decorrelates the data at all spatial scales, but does so without explicit space partitioning, i.e. avoiding tree-like structures or multiresolution filter banks. The task of capturing correlation at many scales is thus undertaken by each individual rank. However, the question of how to partition the coefficients for an effective variable-rate compression is unclear *a priori*. Fortunately, the HOSVD algorithm is guaranteed to produce core slices that are non-increasing in norm [de Lathauwer et al., 2000b]:

$$\sigma_1^{(n)} \geq \sigma_2^{(n)} \geq \ldots \geq \sigma_N^{(n)} \tag{3.3}$$

Furthermore, from the factor matrix orthonormality it follows that the mean square error (MSE) induced by eliminating a coefficient is proportional to its squared magnitude. These facts have been exploited in the past as the basis of several truncation-based HOSVD compression schemes [Wu et al., 2008], [Suter et al., 2013], [Suter et al., 2011b], [Ballester-Ripoll et al., 2015], whereby the least important trailing core slices along each dimension are discarded. However, there are two important aspects that have not been further pursued by these previous approaches. First, although the slice truncation idea is sound as motivated by Eq. 3.3, its granularity is very coarse. Elimination strategies on a coefficient-by-coefficient basis (rather than slice-by-slice) have potential for reducing error. Second (and regardless of the coefficient elimination method chosen), how to encode the surviving coefficients remains an open issue. The apparent usual distribution (Fig. 3.1) invites to use a logarithmic transform. For example, [Suter et al., 2013] use a 9-bit logarithmic quantization scheme: 1 bit for the coefficient sign and 8 for

its magnitude after taking the logarithm of its absolute value. The authors realized the extreme importance of the first element $\mathcal{B}[0,0,0]$ (the *hot corner*) and thus decided to save it separately at full double floating-point precision; this strategy was replicated in other works [Suter et al., 2011b], [Ballester-Ripoll and Pajarola, 2015]. Nevertheless, a truly adaptive compression approach for the HOSVD has not been explored. The strategy we propose generalizes the thresholding-oriented analysis of [Ballester-Ripoll and Pajarola, 2015]: we sort and adaptively partition the set of coefficients according to their relative magnitude, and encode each chunk at a variable quantization resolution.

3.3.2 Further Remarks

Many transformations do not significantly affect the HOSVD. For example, if one permutes some slices of $\mathcal{T}$ along one or more dimensions, its HOSVD will produce the same core $\mathcal{B}$ and factors (with their corresponding rows permuted). Other possible transformations that can be encoded on a HOSVD-compressed data set without modifying $\mathcal{B}$ include translating the data, padding it with zeros, upsampling it with linear interpolation, scaling by a constant, and several others. Many usual data reduction approaches are guaranteed to actually improve HOSVD core sparsity, including downsampling, box-filtered decimation, convolving with any kernel that has band-limited Fourier transform, etc. Discarding the last level of a 3D separable wavelet transform, for instance, is equivalent to zeroing-out 7 octants of the Tucker core.

3.4 Proposed Algorithm

Our compression pipeline consists of four main stages. First, the HOSVD algorithm is run on the N-dimensional data set to yield N orthogonal square factor matrices and an N-dimensional core of the same size as the original. Second, the core coefficients are sorted and partitioned into chunks according to their absolute magnitude; each chunk's coefficients are quantized with a different number q of bits. Third, the presence masks of each chunk are losslessly compressed using run-length encoding (RLE) followed by Huffman encoding. Last, the factor matrices are quantized column-wise, with each column using a different number of bins. All components are immediately streamed upon generation through *zlib*'s lossless compression algorithm before finally writing to disk. The pass through zlib provides a small extra compression (below 5%). See Alg. 1 for a pseudocode of our compression pipeline; its individual building blocks are detailed next.

Algorithm 1 Compress an N-dimensional tensor $\mathcal{T}$ of size $I_1 \times \cdots \times I_N$ at a prescribed mean squared error ϵ.

for $n = 1, \ldots, N$ **do**
  $\mathbf{T}_{(n)} := \text{unfold}(\mathcal{T}, n)$ {Size $I_n \times (I_1 \cdots \widehat{I_n} \cdots I_N)$}
  $\mathbf{T}^{\text{cov}}_{(n)} := \mathbf{T}_{(n)} \cdot \mathbf{T}^T_{(n)}$ {$I_n \times I_n$ covariance matrix}
  $\mathbf{\Lambda}^{(n)}, \mathbf{U}^{(n)} = \text{eig}(\mathbf{T}^{\text{cov}}_{(n)})$ {Full decomposition; eigenvalues $\mathbf{\Lambda}^{(n)}$ in non-increasing order}
  $\mathbf{T}_{(n)} := {\mathbf{U}^{(n)}}^T \cdot \mathbf{T}_{(n)}$ {Right part $\mathbf{\Sigma} \cdot \mathbf{V}^T$ of the SVD}
  $\mathcal{T} := \text{fold}(\mathbf{T}_{(n)})$ {Back to original size}
end for
$\mathbf{a} := \text{flatten}(\mathcal{T})$
$\mathbf{s} := \text{sort}(\text{abs}(\mathbf{a}))$ {Non-decreasing order}
$\mathbf{m} := \text{argsort}(\text{abs}(\mathbf{a}))$ {Indices that map back to the original ordering, i.e. $\mathbf{s}[\mathbf{m}[i]] = \text{abs}(\mathbf{a}[i])$}
$i_0 := 0$ {First threshold}
$\mathcal{M} := \{0, \ldots, I_1 \cdots I_N - 1\}$ {Mask to record coefficients not saved yet}
for $q = 0, \ldots, 63$ **do**
  Find the largest $0 \le i_{q+1} \le I_1 \cdots I_N$ such that

$$\frac{\text{error}(\mathbf{s}[i_q : i_{q+1} - 1])}{i_{q+1} - 1 - i_q} \le \epsilon$$

  where the $\text{error}(\mathbf{u})$ is defined as $\|\mathbf{u} - \text{dequant}(\text{quant}(\mathbf{u}, q), q)\|^2$ {Since error() grows monotonically with i_{q+1}, we use binary search}
  $\mathbf{c} := (0, \ldots, 0)$ {Presence bits: $I_1 \cdots I_N$ zeros}
  $\mathcal{M}^* := \mathcal{M}$
  for each $0 \le j < I_1 \cdots I_N$ such that $i_q \le \mathbf{m}(j) \le i_{q+1}$ **do**
    Save $\text{quant}(\mathbf{a}[j], q)$
    $\mathbf{c}[j] := 1$ {Mark as saved}
    $\mathcal{M}^* := \mathcal{M}^* \setminus j$
  end for
  Save $\text{Huffman}(\text{RLE}(\mathbf{c}[\mathcal{M}]))$
  $\mathcal{M} := \mathcal{M}^*$
  if $\mathcal{M} = \emptyset$ **then**
    Exit loop {The whole core has been encoded}
  end if
end for
if $\mathcal{M} \neq \emptyset$ **then**
  Save remaining coefficients in $\mathcal{M}$ explicitly with 64-bit floating point
end if
for $n = 1, \ldots, N$ **do**
  for $j = 1, \ldots, I_n$ **do**
    Save $\text{quant}(\mathbf{U}^{(n)}[:, j], q_j^{(n)})$ {$q_j^{(n)}$ as in Eq. 3.6}
  end for
end for

3.4.1 HOSVD Transform

In general one can directly compute the left and right singular vectors of each unfolding $\mathbf{T}_{(n)}$ in one run of any standard SVD algorithm. In volume compression, however, we usually have $\mathbf{T}_{(n)} \in \mathbb{R}^{n \times m}$ with $n \ll m$. In such cases it is much more efficient to compute first the uncentered covariance matrix $\mathbf{T}^{\text{cov}}_{(n)} := \mathbf{T}_{(n)} \cdot \mathbf{T}^{T}_{(n)}$ and then obtain all left singular vectors $\mathbf{U}^{(n)}$ from the full eigenvalue decomposition $\mathbf{T}^{\text{cov}}_{(n)} = \mathbf{U}^{(n)}\mathbf{\Lambda}^{(n)}\mathbf{U}^{(n)T}$. Since $\mathbf{T}^{(n)}_{\text{cov}}$ is a real symmetric matrix, its eigenvalue diagonalization always exists and we can use Eigen's specialized `SelfAdjointEigenSolver` class. The remaining rightmost part of the SVD follows from

$$\mathbf{\Sigma}^{(n)}\mathbf{V}^{(n)} = (\mathbf{U}^{(n)})^{-1}\mathbf{T}^{(n)} = \mathbf{U}^{(n)T}\mathbf{T}^{(n)} \tag{3.4}$$

as the factor matrix $\mathbf{U}^{(n)}$ is orthogonal. We have found that one ALS iteration (i.e. the plain HOSVD decomposition) is enough for practical convergence.

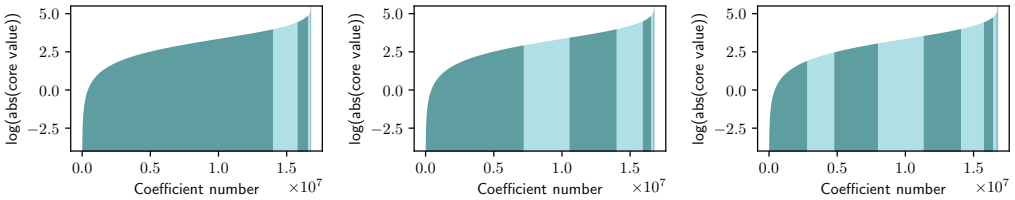
3.4.2 Adaptive Chunk Partitioning

Once the Tucker core $\mathcal{B}$ is available we can turn to our adaptive quantization and encoding scheme. Note that we have not incurred yet any loss of accuracy beyond machine precision, since all ranks are preserved. Our goal now is to produce an approximate core $\widetilde{\mathcal{B}}$ so that each coefficient $\widetilde{\mathcal{B}}[\mathbf{x}] \approx \mathcal{B}[\mathbf{x}]$ contributes equally (on average) to the overall final error. More specifically, we will split up the coefficients in groups (chunks) so that each chunk's total error is proportional to its size and is quantized with a different number of bits q. Let us consider the array $\mathbf{s}$ of core coefficients in absolute value, sorted in ascending order. We have observed (Fig. 3.1) that its elements generally grow at a much faster rate the latter they appear in the sorted magnitude curve. If we were to keep q constant as we move towards the larger elements, the chunk sizes would need to be dramatically reduced in order to keep their overall errors proportional. Having many small chunks is clearly undesirable since they incur in large overhead, namely their respective minimum/maximum bounds plus their presence masks (see next section). To balance this we increase q by 1 after each chunk. The extra bit halves the average per-coefficient error and thus adaptively counteracts the roughly exponential increase in magnitude as we progress along the curve.

Each q-th chunk is encoded using $q + 1$ bits per coefficient, with each first bit storing the sign and the remaining q bits the following linear quantization. Element-wise:

$$\mathbf{s}[j] \mapsto \text{round}\left((2^q - 1) \cdot \frac{\text{abs}(\mathbf{s}[j]) - \text{abs}(\mathbf{s}[i_q])}{\text{abs}(\mathbf{s}[i_{q+1} - 1]) - \text{abs}(\mathbf{s}[i_q])}\right) \tag{3.5}$$

with $\mathrm{abs}(\mathbf{s}[i_q])$ and $\mathrm{abs}(\mathbf{s}[i_{q+1}-1])$ being the first and last elements of the q-th chunk (and therefore, its minimum and maximum bounds), which are stored separately and without loss. The only exception is the first chunk $q = 0$, which uses zero bits overall and does not need a sign (all its elements are mapped to 0). We quantize and save all coefficients within each chunk in the order they appeared in the flattened core tensor (not in their order in $\mathbf{s}$ after sorting by absolute value), so that they can be directly and efficiently set back in place during decompression.



(b) PSNR = 35 (c) PSNR = 50 (d) PSNR = 65

Figure 3.1: *Resulting adaptive chunk sizes along the Foot's sorted core curve at three target PSNR levels. In each plot the leftmost chunk is zeroed-out, while remaining chunks use an increasingly large number q of quantization bins.*

3.4.3 Mask Encoding

The quantization approach discussed above takes care of coefficients' magnitude but disregards their original position in the core. We encode this crucial information separately and without loss, using a vector of presence bits $\mathbf{c}$ and a set $\mathcal{M}$ that keeps coefficient positions that have not yet been quantized. The system works as a sieve: newly processed coefficients at each chunk are marked as ones in $\mathbf{c}$. Then $\mathbf{c}[\mathcal{M}] = \{\mathbf{c}[j] \mid j \in \mathcal{M}\}$ is compressed by RLE and Huffman encoding. Finally, these coefficients are crossed out from the mask $\mathcal{M}$. Thus, the bit streams that need to be compressed after each chunk become progressively smaller.

This chunk partitioning and encoding strategy generalizes both HOSVD truncation [Suter et al., 2013], [Suter et al., 2011b], [Ballester-Ripoll et al., 2015] and hard-thresholding [Ballester-Ripoll and Pajarola, 2015]. The latter defines only three kinds of coefficients: the zeroed-out, the quantized, and the single hot-corner. Instead, the proposed method works with a variable number of threshold-defined chunks, each of which is handled in the same way but with an increasing parameter $q \geq 0$. Note that the size of the final compressed masks can vary in principle if we either a) choose a FORTRAN-style core flattening (instead of C-style), or b) permute the input dimensions in some way. In practice we have found such variations to influence very little the overall compressed file size.

3.4.4 Factor Quantization

The square factors $\{\mathbf{U}^{(n)}\}_n$ usually account for a small proportion of the overall NNZ compared to the tensor core (except for very low bit-rates). We choose to quantize their columns independently, based on the observation that each factor column interacts with exactly one core slice. We quantize the j-th column of the n-th factor using a number of bits q based on the maximal q value of all its interacting core slice coefficients:

$$q_j^{(n)} = \max \left\{ q(\mathcal{B}[\overbrace{:,\ldots,:}^{n-1}, i, \overbrace{:,\ldots,:}^{N-n}]) \right\} + k \tag{3.6}$$

where k is a constant. Conservative values of k are advisable as factor columns are typically much smaller than their corresponding core slices. In our experiments we found $k = 2$ to be the best compromise, and we use this value throughout all measurements.

3.5 Decompression

Decompression follows straightforwardly by inverting the described steps: the core is populated chunk by chunk using Huffman and RLE decoding plus linear dequantization, and then reconstructed via N TTM products. This is significantly faster than compression since a) no covariance and eigenvalue decomposition are needed; and b) no chunk thresholds need to be selected.

3.6 Results

We have tested the proposed method with three 8-bit unsigned int and five 64-bit float data sets (Tab. 3.1).

We have measured the compression performance of TTHRESH against two state-of-the-art compressors, SZ and ZFP. For SZ we use the relative error bound mode and vary the relative bound ratio parameter, while for ZFP we use the fixed accuracy mode and vary its absolute error tolerance. Since these methods do not support integer data types, we first cast all 8-bit volumes to 64-bit floats; we measure compression ratios w.r.t. the original data. Fig. 3.2 shows the resulting error curves in terms of PSNR vs. compression ratio over all sample volumes. In Figs. 3.4 and 3.5 we present several renderings before and after compression with TTHRESH at two levels of quality.

Regarding computational speed, we plot in Fig. 3.3 the compression and decompression times for the smallest data set (the Teapot, 11.1MB) as well as for

Table 3.1: *The 8 data sets used in this chapter.*

Name	Dimensions	Type	Size	Source
Foot	$256 \times 256 \times 256$	8-bit unsigned int	16 MB	The Volume Library [iap,]
Boston teapot (with a lobster)	$256 \times 256 \times 178$	8-bit unsigned int	11.1 MB	The Volume Library [iap,]
Engine	$256 \times 256 \times 256$	8-bit unsigned int	16 MB	The Volume Library [iap,]
Channel pressure	$512 \times 512 \times 512$	64-bit float	512 MB	Johns Hopkins Turbulence Database [jht,]
Viscosity	$384 \times 384 \times 256$	64-bit float	288 MB	Miranda simulation [Cabot and Cook,]
Density	$384 \times 384 \times 256$	64-bit float	288 MB	Miranda simulation [Cabot and Cook,]
U	$288 \times 192 \times 28$	64-bit float	11.8 MB	Community Earth System Model [ces,]
Jet-U	$400 \times 250 \times 200$	64-bit float	152.6 MB	Sandia National Laboratories [Grout et al., 2011]

one of the 512MB ones (the Isotropic-fine). Our method is between 0.5 and 2 orders of magnitude slower than ZFP, although it compares somewhat better against SZ. It is rather asymmetric as we expected (Sec. 3.5). Note also that the varying accuracy curves between all three compared algorithms make a fully fair comparison difficult. For consistency with Fig. 3.2 we do the comparison in terms of time vs. compression ratio, but TTHRESH fares better in terms of quality in large parts of the error spectrum.

3.7 Discussion

We observe that the proposed algorithm achieves very competitive accuracy at medium to high compression ratios and consistently outperforms other compressors at high ratios; see the higher PSNR curves for our method in the middle to right regions of each plot from Fig. 3.2. The overtaking point at which TTHRESH surpasses other algorithms typically produces renderings that are already close to visually indistinguishable to the original data set. This is especially true for higher bit-depths. We believe the method is thus a good choice for applications with significant error tolerance (chiefly, visualization-related). Since each threshold is chosen on a coefficient by coefficient basis, the range of possible final errors has an extremely high resolution. Also, the error that arises from the core compression is upper-bounded thanks to the careful selection of inter-chunk limits within the coefficient curve. The compressed-domain filtering and resampling features are rather unique strengths of the tensor decomposition framework, only possible thanks to its multilinearity. Any separable filter and resampling can be applied at

little cost by manipulating the factors column-wise after dequantization and before the final Tucker reconstruction. This is often much more challenging in other compression methods, especially non-transform ones. Last, even though Lanczos results are visually superior when lowering a data set's resolution, we believe the other decimation methods remain useful for e.g. region selection, projections, etc.

Limitations

As discussed before, TTHRESH's favorable compression rates come at the price of its monolithic approach to the whole transform core. This makes it significantly slower than the other tested methods. Random-access decompression is relatively costly, as one must traverse the whole core in all cases. For example, while progressive transmission is possible in principle (it suffices to send the smallest chunks first), a full decompression of each partial core is still required. We have developed TTHRESH focusing primarily on data reduction rates, and less so on general compression/decompression speed. We have realized that these speeds (especially compression) can be notably increased at a relatively small accuracy cost, for example by adding a core truncation step during the HOSVD, or by early-stopping the binary search that determines the chunk sizes. This will be the subject of future investigation.

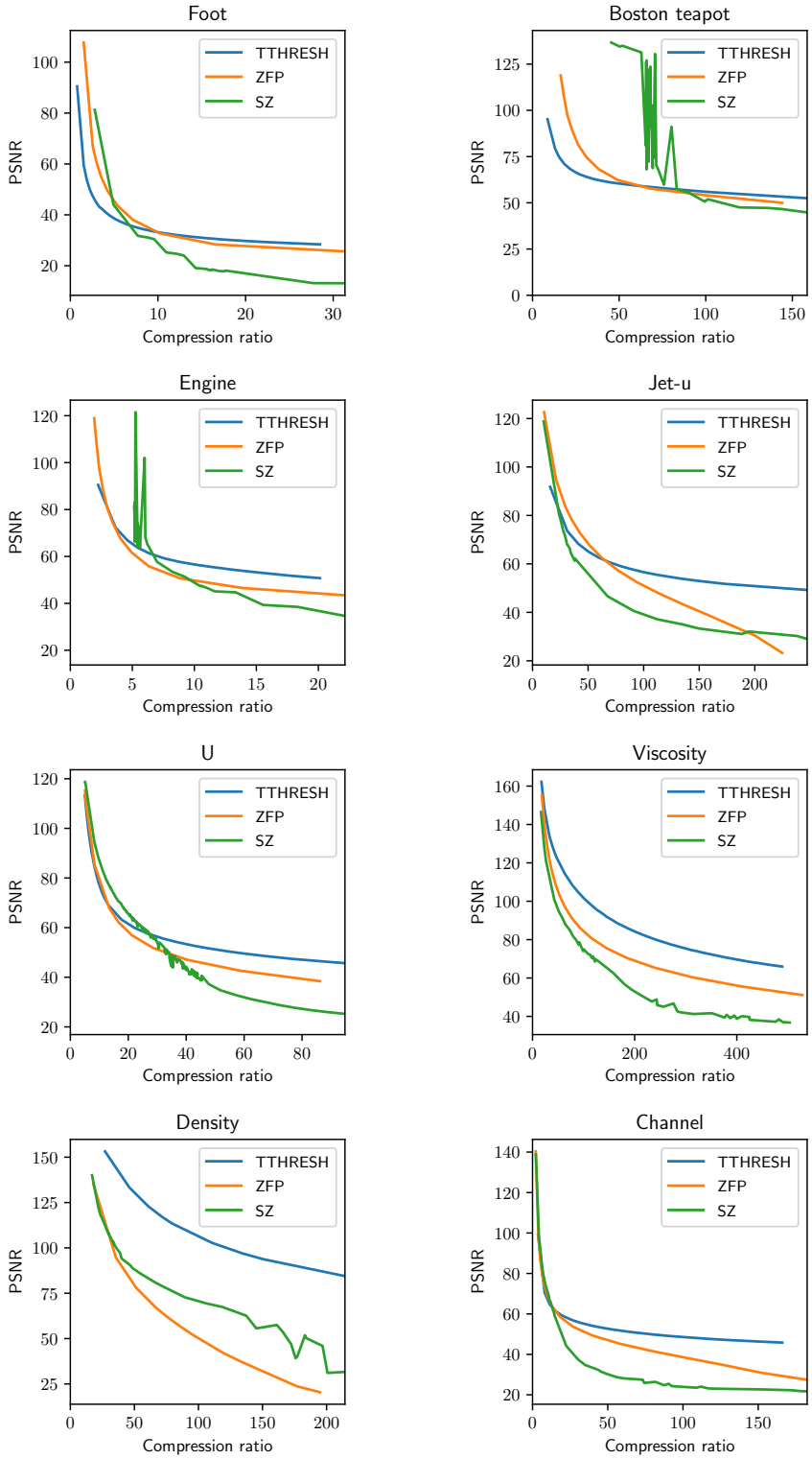


Figure 3.2: *Compression quality curves (higher is better) for our method compared to* SZ *and* ZFP *over 8 example volumes.*

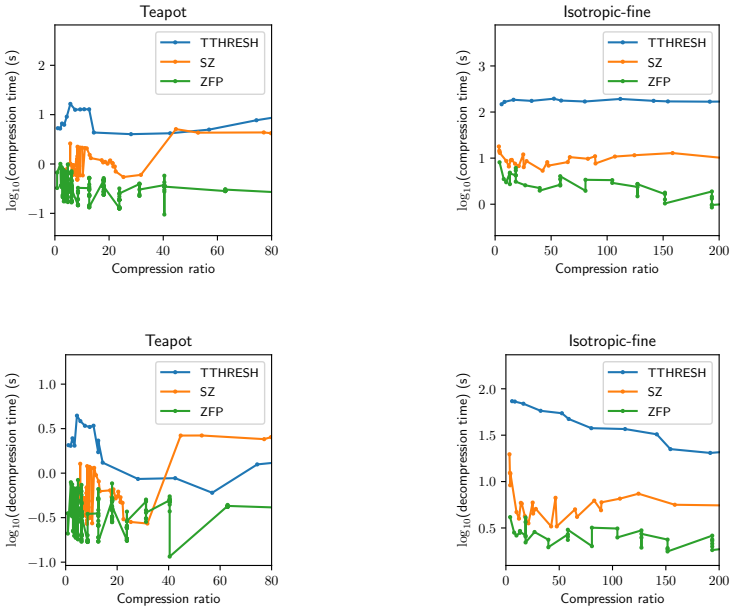


Figure 3.3: *Compression (top row) and decompression (bottom row) times for two volumes and a range of different compression ratios.*

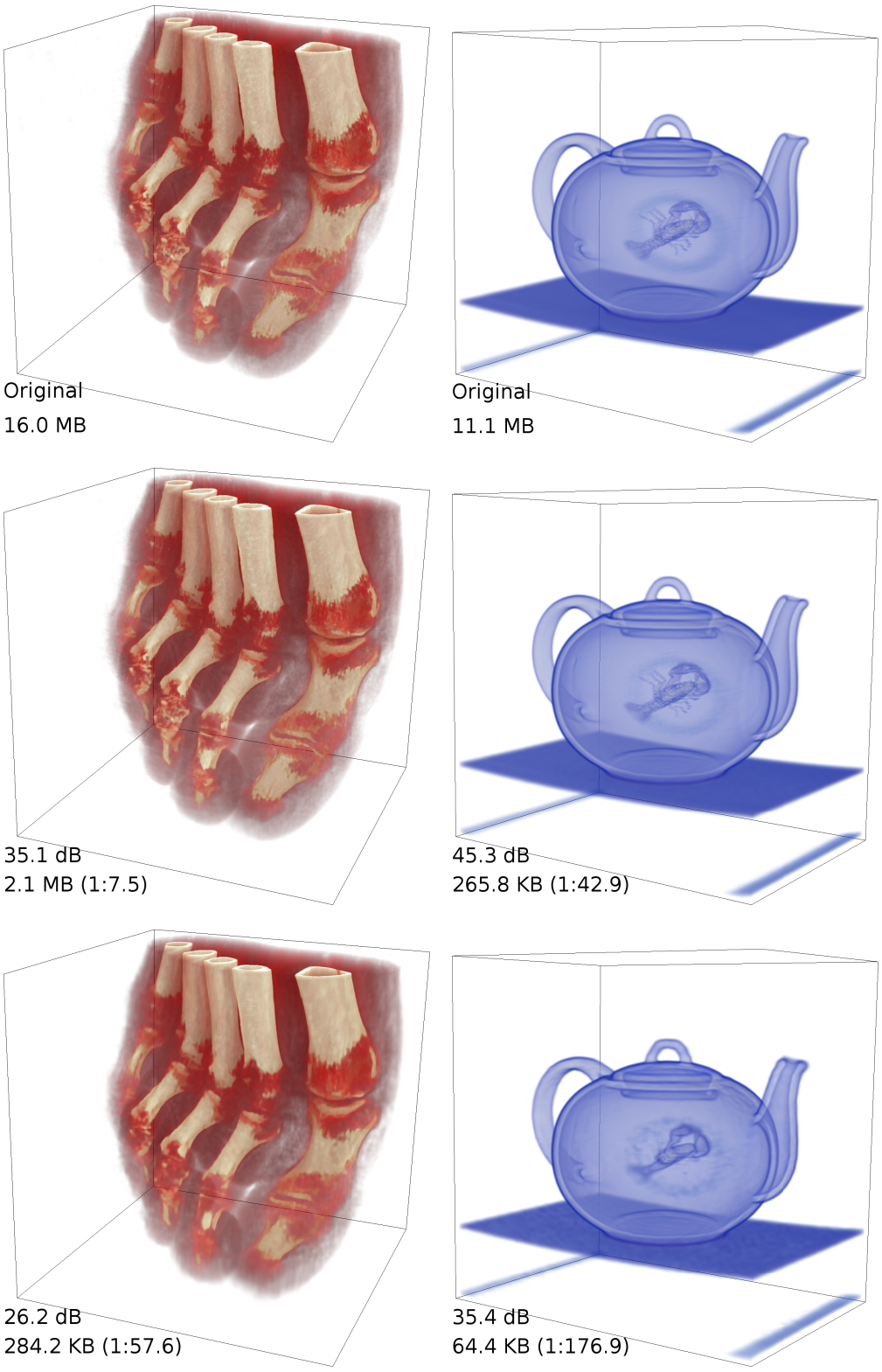


Figure 3.4: *Two example volumes (I): the Foot CT scan and the Boston teapot. Rows from top to bottom: original, higher quality, and lower quality.*

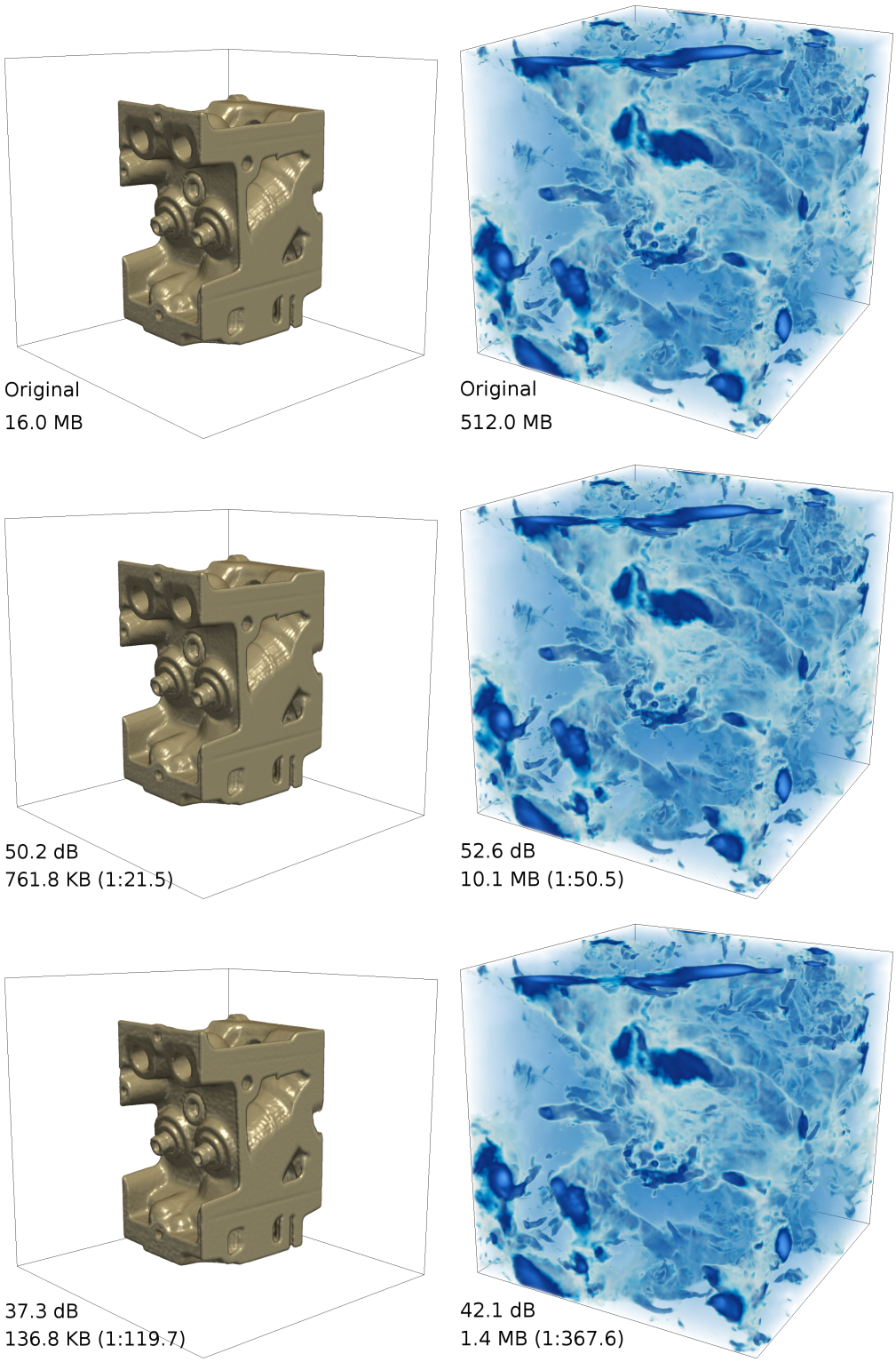
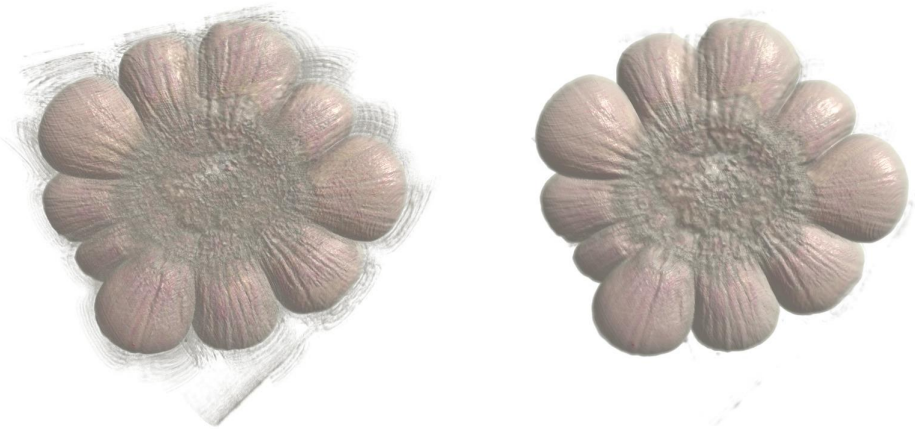


Figure 3.5: *Two example volumes (II): the Engine and the Channel turbulence simulation.*

C H A P T E R 4

MULTIRESOLUTION FILTERING



4.1 Overview

Modern interactive large-scale volume rendering systems typically rely on compressed and out-of-core multiresolution data structures [Balsa Rodríguez et al., 2014]. Such systems precompute volume subregions at all possible resolutions (levels of detail, LOD), also known as *octree bricks*. During visualization they keep bricks of different LODs in GPU memory at once. Besides interactive visual exploration, more advanced data analysis tasks require additional data selection and processing operations. Many signal processing operations, for example filtering, are expensive and challenging to apply across the variable LOD hierarchy. Filter operations are best integrated into the visualization application, and if possible performed efficiently in a compressed form. In this chapter we develop and demonstrate multiresolution filtering in an octree-based visualization system using tensor decomposition. We keep the multiresolution volume data in a memory-efficient form until the latest possible stage before rendering on the GPU (Fig. 4.1). Furthermore, we assume that in addition to the cost for loading compressed volume data from disk to main and into GPU memory, also the cost for decoding –preferably on the GPU just before display– is intrinsic to most multiresolution volume visualization systems, as are the costs for LOD selection and rendering.

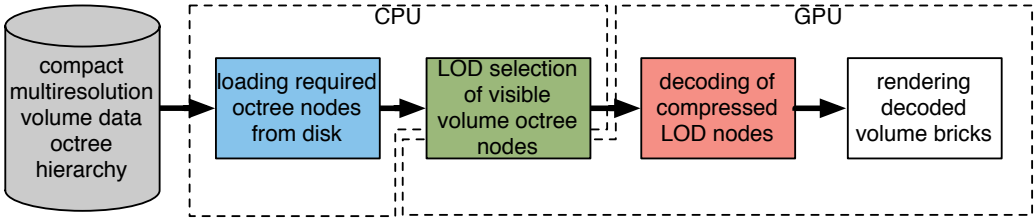


Figure 4.1: *Typical interactive large-scale multiresolution compressed volume visualization pipeline.*

Contribution

We show [Ballester-Ripoll et al., 2017] how a multiresolution tensor-compressed volume representation can support efficient filter operations, and to do so seamlessly across variable reconstruction resolutions. Our solution applies convolution operations directly on the decomposed volumes. This approach generates faster and more accurate filtering results on the selected visible and variable LOD volume bricks than applying the filter to the decoded raw bricks at different resolutions. Furthermore, we integrate this algorithm into a state-of-the-art interactive visualization application, fully compatible with out-of-core multiresolution vol-

ume rendering. The entire factor matrix convolution filtering, downsampling and brick reconstruction process exploits GPU massive parallelism and run at interactive rates for large volumes.

4.2 Background

Large-scale multiresolution visualization systems must support fast variable LOD access and rendering. For instance, [Hadwiger et al., 2012a] propose a rendering architecture for dense and anisotropic datasets obtained by stitching high-resolution 2D tiles. The data is downsampled from its original full resolution in real time. Filtering operations beyond anti-aliasing filters (noise removal, edge detection, etc.) are fundamental in several visualization settings, see for example [Jeong et al., 2009] and [Treib et al., 2012]. See also [Soltészová et al., 2017] for a completely visualization-driven approach, where only voxels whose filtering will produce a significant pixel difference are processed in the first place. Brick boundaries constitute an additional problem in octree-like hierarchies. [Sicat et al., 2014] describe a 3D multiresolution system based on sparse probability density functions that enables consistent and smooth low-pass filtering at multiple resolution levels. However, their representation is not smaller than the input volumes, and only low-pass filter operations for anti-aliasing are demonstrated.

Fourier, discrete cosine or wavelet transforms (FT, DCT, WT respectively) have been used in the past for efficient interactive multiresolution volume visualization [Yeo and Liu, 1995; Grosso et al., 1996; Lippert et al., 1997], also for managing large-scale out-of-core volume data [Rodler, 1999; Ihm and Park, 1999]. More recently, novel approaches like vector quantization, tensor decomposition or sparse coding have successfully been proposed for large multiresolution volume rendering, see e.g. [Guthe et al., 2002; Schneider and Westermann, 2003; Fout and Ma, 2007; Suter et al., 2011b; Gobbetti et al., 2012; Suter et al., 2013] as well as the survey [Balsa Rodríguez et al., 2014]. While vector quantization and sparse coding do not lend themselves well to efficient filtering, transform-based compression like FT, DCT and WT are better suited for signal processing operations. Based on a compressed WT volume representation, [Treib et al., 2012] decompress and evaluate turbulence properties of velocity-vector time-varying volumes during rendering on the GPU. However, that approach is designed for datasets fully fitting into GPU memory at each frame and filters at the full original resolution only.

Overall, while compression domain volume rendering is a well-explored and established paradigm, the problem of efficient filtering over an out-of-core compressed LOD hierarchy has not yet been properly addressed in the context of multiresolution interactive visualization. Tab. 4.1 summarizes several possible visual-

ization pipelines, including the proposed one.

Table 4.1: *Visualization pipelines for several approaches: O = uncompressed data in a multiresolution octree format; OC = octree with compressed volume data; C = compressed full resolution data (no LOD); D = decompression to full resolution; DL = decompression to target LOD; L = downsample to target LOD; F = filter; F* = filter in the compressed domain (much faster than F).*

Method	Observations	RAM	GPU
Multiresolution visualization	Uncompressed data must fit in GPU; artifacts when filtering	O	F
[Gobbetti et al., 2012], also [Suter et al., 2013]	No filtering	OC;	DL
[Treib et al., 2012]	No LOD: full resolution data must fit in GPU	C;	D; F
Filter after decompression before downsampling	Must filter full resolution data even at low LODs	OC;	D; F; L
Filter after downsampling	Slower than our method; artifacts when filtering	OC;	DL; F
Ours	Accurate LOD filtering at any resolution	OC;	F*; DL

4.2.1 Multiresolution Filtering

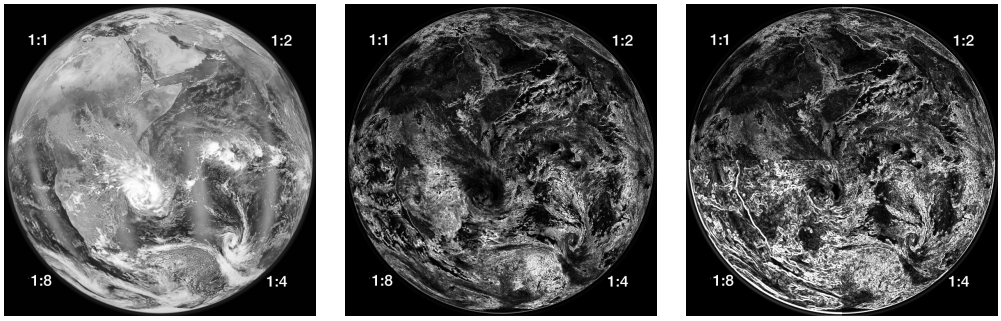
According to the sampling theorem in digital signal processing, downsampling a signal by a factor k will result in aliasing whenever its frequency spectrum is wider than a band of size $2\pi/k$. It is challenging to filter large volume data in an interactive LOD based rendering system, since different downsampling levels demand filtering at various downsampled resolutions. More specifically, three possible options for compression-based multiresolution filtering can be considered:

1. *Filtering after decompression before downsampling (FD).* The most accurate reduced resolution filtered result can be achieved by filtering the original high resolution signal with the unmodified filter kernel, followed by downsampling the result. This is computationally prohibitive since it requires decompression and filtering of the full resolution data, even when only a lower resolution LOD is actually needed for visualization. This approach defeats the purpose of using an octree representation altogether, since only leaf nodes (i.e. full resolution bricks) are ever read.

2. *Filtering after decompression and downsampling (DF).* If the input signal $\mathcal{T}$ is downsampled by a factor 2^f, also the filter kernel $\mathcal{G}$ should equally be downsampled before convolution. In addition to losing detailed filter

responses, some compact filters simply cannot be downsampled at all because the resulting filter mask size $K/2^f$ would be too small to capture the essential kernel properties (e.g. $K = 3$ and $f > 1$). In this cases, this strategy must resort to applying the unmodified kernel to variable LOD regions, which entails incorrect results. See Fig. 4.2 for a comparison of FD vs. DF on a sample image.

3. *Tensor-approximated filtering (TAF).* Filtering can be performed in a cost-efficient way in a transformed and compressed domain, yet ensuring the filter operator's accurate full resolution response. The data to be visualized can directly be reconstructed at the desired spatial resolution and LOD after filtering. This approach is the only one in this context that can cope with both large data sizes and several resolution levels at once.



(a) No filtering (b) FD (c) DF

Figure 4.2: *(a) Image with variable resolution quadrants, subsampling from 1:1 down to 1:8. Sobel operator applied (b) before downsampling (FD) and (c) after downsampling (DF). FD shows a much smoother response across resolution discontinuities.*

4.3 Octree Tucker Decomposition

A multiresolution hierarchy (without support for filtering) was defined and combined with Tucker compression in [Suter et al., 2013]. Given an input volume octree, their algorithm computes (Fig. 4.3):

- Three global basis factor matrices $\mathbf{U}^{(n)}$ with row dimensions I_n equal to the input size and R_n columns for $n = 1, 2, 3$
- A small core tensor $\mathcal{B}_{ijk} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ for each octree node, obtained by truncating the HOSVD factors [de Lathauwer et al., 2000b]

A volume brick B_{ijk} on level l can be reconstructed as $\mathcal{B}_{ijk} \times_1 {}^{\downarrow l}\mathbf{U}^{(1)}_{J_1^i} \times_2 {}^{\downarrow l}\mathbf{U}^{(2)}_{J_2^j} \times_3 {}^{\downarrow l}\mathbf{U}^{(3)}_{J_3^k}$, with the factor matrix row ranges $J_n^{\{i,j,k\}}$ corresponding to the brick's spatial location; see also 4.4. Note that for bricks on level l, the global factor matrices are first subsampled by the factor 2^{H-l} in the row dimensions, indicated by ${}^{\downarrow l}\mathbf{U}^{(n)}$, before reconstruction. Assuming a volume of size I^3 and octree bricks of size B^3, setting the core tensor dimension to $R = B/2$ results in a compact format requiring only a) three thin global factor matrices with $3I \cdot R$ elements in total; and b) 2^{3l} core bricks of size $B^3/8$ at each level l, amounting (geometric series) to a total of $\approx I^3/7$ core elements across all octree levels.

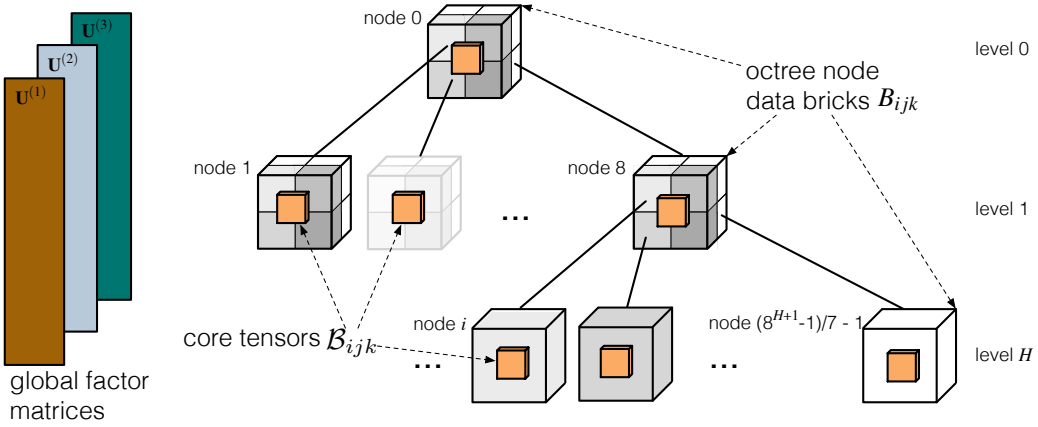


Figure 4.3: *Our system and [Suter et al., 2013] keep three global factor matrices and one small core tensor for each octree node.*

We adopt the hierarchy from [Suter et al., 2013] with one key alteration: we always work with the full resolution global factor matrices, which allows multi-scale filtering in the compressed domain. This important modification preserves the original spatial resolution information, a critical property as we will show in Sec. 4.5.

4.4 Tensor Compressed Domain Filtering

Tensor decompositions allow for easy basis manipulation owing to their multi-linearity (Sec. 2.7), and the Tucker format is no exception. Orthonormal Tucker factors provide a convenient and immediate bijection between any tensor and its transform. Thus, each factor row maps one-to-one onto a slice (or hyperslice) of the input data set. Furthermore, linear combinations of rows produce linear combinations of the corresponding slices (Fig. 4.4).

For example, we can convolve any Tucker-compressed tensor by a separable kernel $\mathcal{G}[\mathbf{x}] = \mathbf{g}^{(1)}[x_1] \cdot \mathbf{g}^{(2)}[x_2] \cdot \mathbf{g}^{(3)}[x_3]$ by column-wise factor convolution

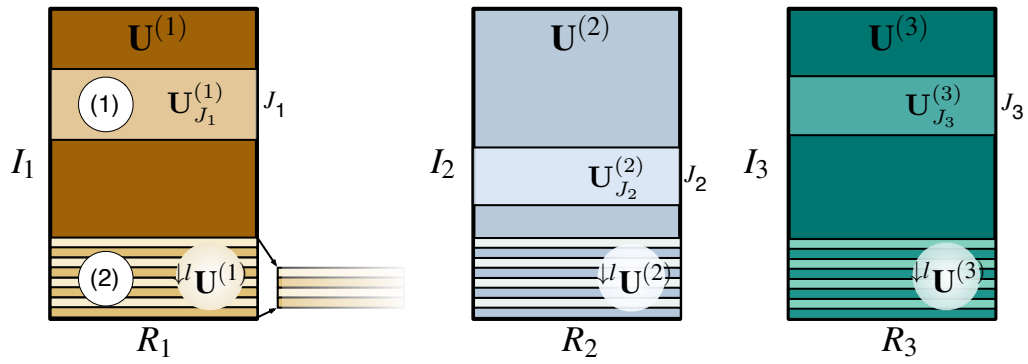


Figure 4.4: *(1) Factor matrix row subranges $\mathbf{U}^{(n)}_{J_n}$ allow for spatial selection of a sub-volume $J_1 \times J_2 \times J_3$. (2) Downsampled factor matrices $^{\downarrow k}\mathbf{U}^{(n)}$ reconstruct to lower resolution data.*

(Fig. 4.5):

$$\widetilde{\mathcal{T}} * \mathcal{G} = \mathcal{B} \times_1 (\mathbf{U}^{(1)} * G_1) \times_2 (\mathbf{U}^{(2)} * G_2) \times_3 (\mathbf{U}^{(3)} * G_3)$$

Furthermore, we can approximate non-separable 3D convolution kernels as a sum of S rank-1 tensors $\mathcal{G} \approx \widetilde{\mathcal{G}} = \sum_{s=1}^{S} \mathbf{g}_s^{(1)} \otimes \mathbf{g}_s^{(2)} \otimes \mathbf{g}_s^{(3)}$ (CP decomposition). Each filter factor column can be applied in the compressed Tucker domain by accumulating rank-1 filter convolutions:

$$\widetilde{\mathcal{T}} * \widetilde{\mathcal{G}} = \sum_{s=1}^{S} \mathcal{B} \times_1 (\mathbf{U}^{(1)} * \mathbf{g}_s^{(1)}) \times_2 (\mathbf{U}^{(2)} * \mathbf{g}_s^{(2)}) \times_3 (\mathbf{U}^{(3)} * \mathbf{g}_s^{(3)}) \tag{4.1}$$

In other words, instead of convolving a 3D filter with the volume data in the spatial domain, we can apply S 1D convolutions on the factor matrices in the tensor compressed domain. A rank-S filter of size K^3 directly applied on a volume of size I^3 in the spatial domain would amount to $\mathrm{O}(I^3K^3)$ operations, while the application of the decomposed filter on the factor matrices, i.e. all $\mathbf{U}^{(n)} * \mathbf{g}_s^{(n)}$, only costs $3 \cdot \mathrm{O}(IKRS)$. Now, assume we partition the input I^3 volume into bricks of size B^3, and each is compressed with R Tucker ranks. Note that R and S are rather small numbers, with $R = B/2 \ll I$ and $S < K$, often even $S \leq 3$. Given a per-brick reconstruction cost of $\mathrm{O}(B^3R)$, a full reconstruction amounts to $\mathrm{O}(I^3R)$ with $R < B \ll I$ in total for the finest level of resolution. Therefore, the volume reconstruction greatly dominates the compressed domain filtering cost. Nonetheless, in a compressed data visualization system, the decompression cost is an inherent part of the rendering algorithm itself and has to be performed anyway.

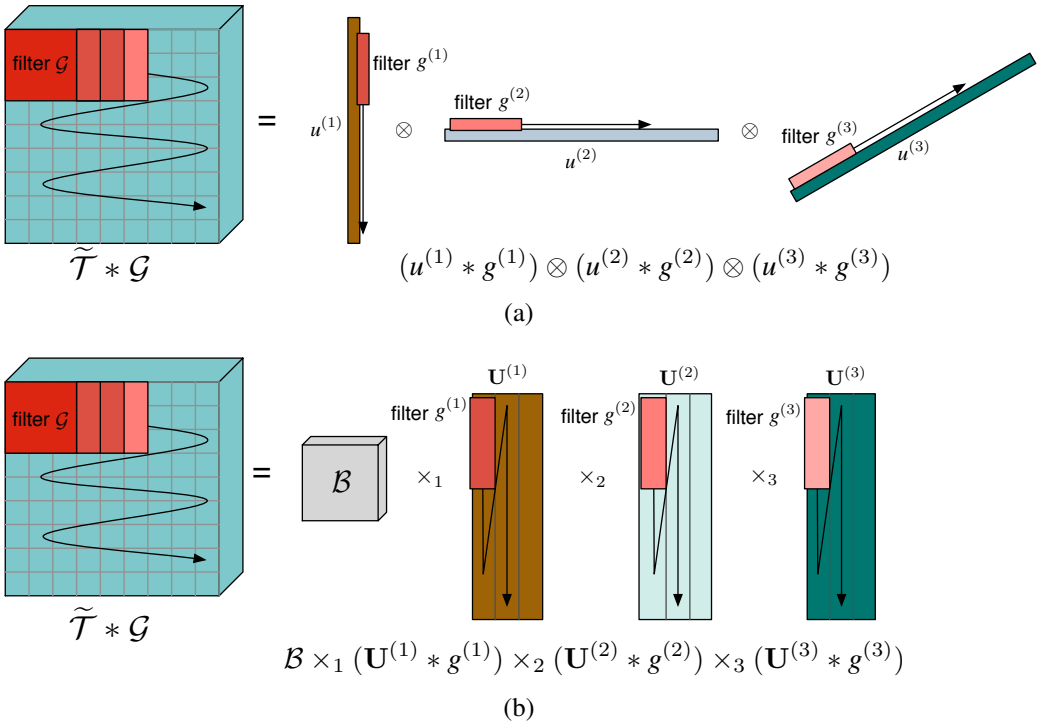


Figure 4.5: *Separable convolution filter $\mathcal{G}$ applied to a 3D CP (a) and Tucker (b) tensor decomposition.*

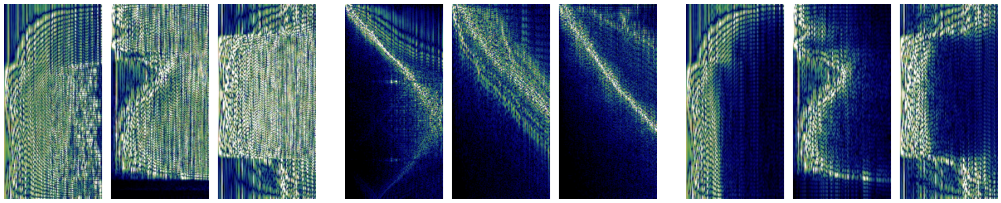
Thus, tensor decomposition domain filtering is not an overhead introduced on top of compression domain volume rendering but rather a cost reduction compared to direct volume filtering.

Note that besides filter convolutions, other frequency domain manipulation techniques are possible in the tensor domain. For example, one can apply any band-pass filter by i) computing the DCT column-wise on the factor matrices; ii) removing the desired frequencies; iii) performing the inverse DCT; and iv) reconstructing (see Fig. 4.6).

4.5 Proposed Multiresolution Filtering

4.5.1 Overview

The basic framework described above in Sec. 4.4 allows tensor compressed domain filtering as suggested in option 3 (TAF). For online multiresolution filtering, we first apply the convolution filter $\widetilde{\mathcal{G}}$ on the full resolution global factor matrices to obtain $\mathbf{U}_*^{(n)}$ and then downsample them by the factor 2^{H-l} for octree height H



(a) Original (b) Column-wise DCT (c) Result after filtering

Figure 4.6: *Left: Tucker factor matrices of the Bonsai CT volume (256^3, compressed to 128 ranks). Center: discrete column-wise cosine transform. Right: low-pass filtered result after keeping only the lowest 20% DCT frequency components and converting back from frequency domain (equivalent to convolution with a sinc kernel). The color map acts on the absolute values; the Tucker core remains always unchanged.*

and target resolution level l. The resulting matrices ${}^{\downarrow l}\mathbf{U}_*^{(n)}$ are then used for the reconstruction of the filtered volume as illustrated in Fig. 4.7(b), with each brick using a *matrix chunk*.

If the columns of each n-th Tucker factor span the leading left singular vectors of a volume's n-th unfolding, then progressive reconstruction is possible, i.e. quality degrades smoothly as we discard rightmost columns. To guarantee this in our multiresolution system, we keep one copy of the global factor matrices for each octree level and re-orthogonalize their chunks accordingly during preprocessing, see also Fig. 4.7(a). For example, for the first level below the octree root corresponding to $2\times2\times2$ nodes, the factor matrices are vertically split in half $\mathbf{U}^{(n)} \rightarrow \{\mathbf{U}^{(n)}_{J^1_n}, \mathbf{U}^{(n)}_{J^2_n}\}$, each part is re-orthogonalized $\mathbf{U}^{(n)}_{J^i_n} \rightarrow \mathbf{U}^{'(n)}_{J^i_n}$, and finally stitched together again into one tall matrix $\mathbf{U}^{'(n)}_{J^1_n} \cup \mathbf{U}^{'(n)}_{J^2_n} \rightarrow \mathbf{U}^{'(n)}$.

During interactive visualization, after selecting a convolution filter and processing the factor matrices as outlined above, the displayed LOD octree node bricks must all be updated and reconstructed according to Eq. 4.1. Each brick requires three specific filtered and downsampled chunks. For example, a third level brick B_{ijk} would be reconstructed using its core tensor $\mathcal{B}_{ijk}$, from octree level $l=3$, and the matching rows $J_n^{\{i,j,k\}}$ of the corresponding filtered third level matrices ${}^{\downarrow 3}\mathbf{U}_*^{'''(n)}$.

We handle inter-brick border artifacts and discontinuities by incorporating a number m of overlapping border rows when splitting the global factor matrices vertically into 2^l segments for octree level l. In other words, each core encodes a brick plus certain margins. This allows for correctly convolving with kernel sizes up to $2m$. These extra rows do not cause additional reconstruction cost as they are only needed to define the tensor decomposition and for filtering the basis factors.

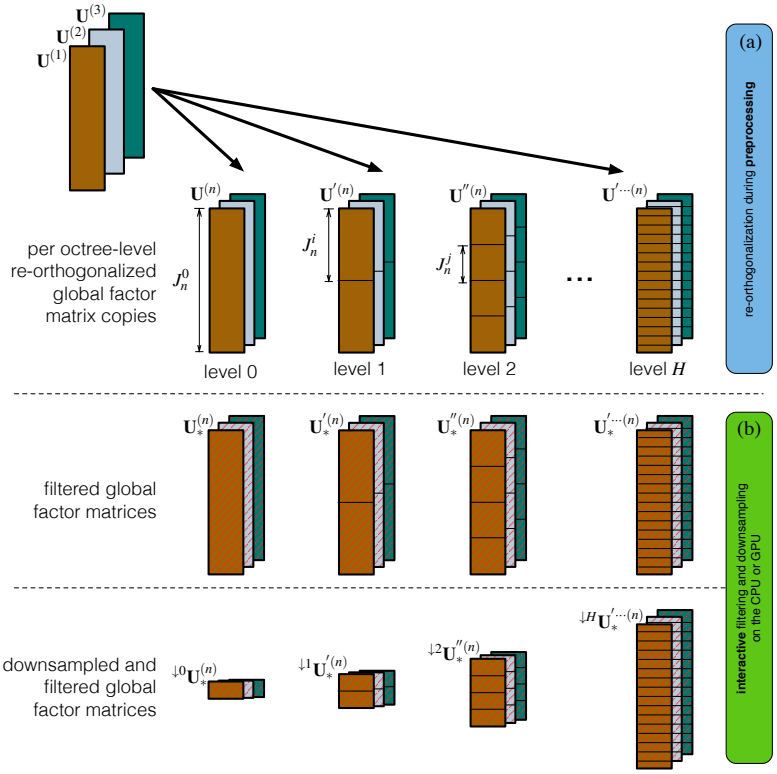


Figure 4.7: *(a) Chunk-wise basis factor matrix re-orthogonalization for each octree level during the pre-process. (b) Filtering and downsampling of the factor matrices at all levels at run-time.*

4.5.2 Decomposition Stage

For simplicity in the exposition we assume a regular input volume of size I^3. Our system uses volume bricks of size B^3 with $B = 64$ for good performance. The global Tucker decomposition factor matrices are of size $I \times R$ with $R = B/2 = 32$ and the octree node core tensors have thus $R^3 = B^3/8$ elements each. The convolution filters are of variable size K, and if non-separable will be approximated by a rank-S CP tensor decomposition, usually only requiring $S \leq 3$.

During the pre-process stage, the input volume $\mathcal{T}$ is decomposed into three learned global factor matrices $\mathbf{U}^{(n)}$ using an ALS algorithm (App. A.1.2) with 3 iterations per mode. The octree hierarchy of core tensors has height $\log_2(I/B)+1$. The implementation follows the outline given in [Suter et al., 2013], especially with respect to using memory mapping and parallelization for handling large input volume datasets. A key difference, however, is that the factor matrices are

Algorithm 2 Building an H-level tensor octree from input volume $\mathcal{T}$.

1: $[[\mathcal{B}, \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \mathbf{U}^{(3)}]] :=$ TUCKER($\mathcal{T}$)
2: **for** $l = 1, \ldots, H$ **do**
3: **for** $i = 1, \ldots I_1/B$ **do**
4: $\mathrm{chunk}(\mathbf{U}'^{\cdots(1)}, i) := \mathrm{LSV}(\mathrm{chunk}(\mathbf{U}'^{\cdots(1)}, i))$
5: **end for**
6: **for** $j = 1, \ldots I_2/B$ **do**
7: $\mathrm{chunk}(\mathbf{U}'^{\cdots(2)}, j) := \mathrm{LSV}(\mathrm{chunk}(\mathbf{U}'^{\cdots(2)}, j))$
8: **end for**
9: **for** $k = 1, \ldots I_3/B$ **do**
10: $\mathrm{chunk}(\mathbf{U}'^{\cdots(3)}, k) := \mathrm{LSV}(\mathrm{chunk}(\mathbf{U}'^{\cdots(3)}, k))$
11: **end for**
12: **for all** $i = 1, \ldots I_1/B, j = 1, \ldots I_2/B, k = 1, \ldots I_3/B$ **do**
13: $\mathcal{B}'^{\cdots}_{ijk} := \mathcal{T}'^{\cdots}_{ijk} \times_1 \mathrm{chunk}(\mathbf{U}'^{\cdots(1)}, i)^T \times_2 \mathrm{chunk}(\mathbf{U}'^{\cdots(2)}, j)^T \times_3 \mathrm{chunk}(\mathbf{U}'^{\cdots(3)}, k)^T$
14: **end for**
15: **end for**

not stored downsampled for each octree level. Instead, we keep H full resolution copies; each copy is separately re-orthogonalized into 2^l chunks of 2^{H-l} consecutive rows for each octree level l as outlined in the previous section. Such copies are critical to preserve the full spatial information at every octree level.

Alg. 2 shows the modified pre-process with the routine $\mathrm{chunk}()$ selecting the 2^{H-l} factor matrix rows for the i, j or k-th chunks, and $\mathrm{LSV}()$ performing the re-orthogonalization over the selected rows by taking the R leftmost singular vectors from the SVD. Note that in this step the overlapping border rows are also incorporated but not explicitly shown for simplicity. Each Tucker core is finally computed by projecting the corresponding input volume brick at the right resolution, denoted by $\mathcal{T}_{ijk}$, onto the basis factor matrices.

4.5.3 Basis Factor Matrix Filtering

Whenever a new convolution filter is selected at run-time, the global factor matrices on the different octree levels are all convolved column-wise by the appropriate filter vector as $\mathbf{U}_*'^{\cdots(n)} \leftarrow \mathbf{U}'^{\cdots(n)} * \mathbf{g}^{(n)}$. The visible volume bricks have then to be reconstructed (on the GPU) before they can be rendered again. The filter may be selected from a predefined set, constructed using an editor, or discretized from an analytic expression. The filtering of the basis factor matrices, which takes a very small portion of the total time (see also Sec. 4.6), is also performed in parallel on

the GPU.

The user can apply any separable filter directly, but if a generic filter $\mathcal{G}$ is given, a (very) low rank-S CP decomposition $\sum_{s=1}^{S} \mathbf{g}_s^{(1)} \circ \mathbf{g}_s^{(2)} \circ \mathbf{g}_s^{(3)}$ must be first obtained, e.g. with ALS. Prominent examples of this category include the Laplacian of Gaussian, unsharp masking, and the Difference of Gaussians (DOG). For commonly used filters, their low-rank CP decomposition can be precomputed and readily provided to the user. Given the decomposed filter kernel, the visible volume bricks can then be reconstructed as in Eq. 4.1 and described in Alg. 3.

More complex operators may even rely on non-linear combinations of simple convolution steps. For instance the Sobel operator, an example of fast edge detection, is computed in 3D as

$$\sqrt{(\mathcal{T} * \mathcal{G}_{\mathrm{x}})^2 + (\mathcal{T} * \mathcal{G}_{\mathrm{y}})^2 + (\mathcal{T} * \mathcal{G}_{\mathrm{z}})^2} \tag{4.2}$$

where $\mathcal{G}_x, \mathcal{G}_y$, and $\mathcal{G}_z$ are gradient-estimating kernels along each spatial axis. They are all separable individually but their overall aggregated filter effect is not.

4.5.4 Reconstruction

Bricks must be newly decompressed when either the LOD in the corresponding octree nodes changes or a new filter operation is requested during interactive visualization. LOD-based updates have been extensively discussed and demonstrated in many interactive multiresolution volume rendering approaches and we can focus primarily on the filtering aspect. Alg. 3 outlines the filtering and reconstruction steps involved when a brick B_{ijk} needs to be updated. First, the filter's 1D basis components $\mathbf{g}^{(n)}$ are convolved column-wise with the basis factor matrices, thus $\mathbf{U}^{\prime\cdots(n)} \rightarrow \mathbf{U}_*^{\prime\cdots(n)}$, computed over row segment ranges $J_n^{\{i,j,k\}}$. Second, the required $\mathrm{downsample}()$ is performed on these factor matrix segments for each given level l, hence $\mathbf{U}_*^{\prime\cdots(n)} \rightarrow {}^{\downarrow l}\mathbf{U}_*^{\prime\cdots(n)}$. These first steps, lines 3 to 5 in Alg. 3, are in fact not carried out for each single brick individually but are all done in one go. With the filtered and downsampled factor matrices, the brick B_{ijk} is eventually reconstructed. If the filter is represented by a rank-S CP decomposition, this process is iterated S times and the results accumulated in B_{ijk}. As illustrated in Fig. 4.8, all crucial computational steps (filtering, downsampling and reconstructing) are performed on the GPU.

4.5.5 Rendering

The view-dependent LOD based volume rendering follows the principles of the *TAMRESH* system [Suter et al., 2013] and is outlined in Fig. 4.8. For each rendered frame, the multiresolution octree representation is traversed and the required

Algorithm 3 Reconstructing a volume brick B_{ijk} from the level l, while applying a rank-S CP-decomposed convolution filter $\mathcal{G} \approx \sum_{s=1}^{S} \mathbf{g}_s^{(1)} \circ \mathbf{g}_s^{(3)} \circ \mathbf{g}_s^{(3)}$.

$B_{ijk} := 0$
for $s = 1, \dots S$ **do**
  $^{\downarrow l}\mathbf{U}_*^{'\cdots(1)} := \text{downsample}(\mathbf{U}^{'\cdots(1)} * \mathbf{g}_s^{(1)})$
  $^{\downarrow l}\mathbf{U}_*^{'\cdots(2)} := \text{downsample}(\mathbf{U}^{'\cdots(2)} * \mathbf{g}_s^{(2)})$
  $^{\downarrow l}\mathbf{U}_*^{'\cdots(3)} := \text{downsample}(\mathbf{U}^{'\cdots(3)} * \mathbf{g}_s^{(3)})$
  for all $i = 1, \dots I_1/B, j = 1, \dots I_2/B, k = 1, \dots I_3/B$ **do**
    $B_{ijk} := B_{ijk} + \mathcal{B}_{ijk} \times_1 \text{chunk}(^{\downarrow l}\mathbf{U}_*^{'\cdots(1)}, i) \times_2$
    $\text{chunk}(^{\downarrow l}\mathbf{U}_*^{'\cdots(2)}, j) \times_3 \text{chunk}(^{\downarrow l}\mathbf{U}_*^{'\cdots(3)}, k)$
  end for
end for
return B_{ijk}

LOD nodes are selected and rendered using ray-casting. The LOD selection and rendering mechanism exploits pooling and caching of bricks between rendering frames. Therefore, bricks only have to be loaded and reconstructed if they are not already cached from an earlier displayed frame.

4.6 Results

Note: the following experimental measurements, volume renderings and supporting text were produced and published [Ballester-Ripoll et al., 2017] in joint work with David Steiner from the Visualization and MultiMedia Lab. The resulting report is here reproduced from the manuscript for completeness, and its attribution is shared by all coauthors.

4.6.1 Software and Hardware Used

Our system is implemented in C++/OpenGL on top of *Equalizer* [Eilemann et al., 2009], a scalable parallel rendering framework for OpenGL-based applications. For its convenience in graphics and visualization we use the C++ library *vmmlib* [vmm, c], which uses BLAS and LAPACK parallel routines to optimize tensor operations including Tucker decomposition and reconstruction. We extended the parallel GPU-based volume ray-casting application described in [Suter et al., 2013] to incorporate our modified Tucker bases and filtering process. Both Tucker brick reconstruction and matrix downsampling/filtering were implemented using CUDA, while octree traversal and core uploading is undertaken by the CPU. All

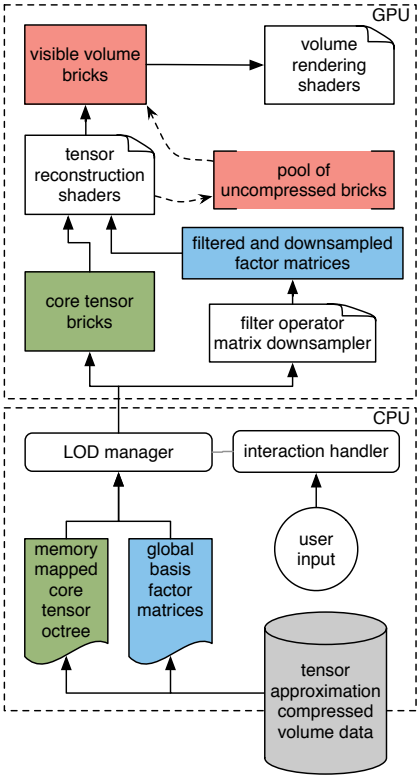


Figure 4.8: *Overview of the proposed multiresolution volume filtering and rendering system.*

such updates are asynchronous. Octree decomposition was performed on the CPU using vmmlib [vmm, c] using OpenMP parallelization. Our results have been obtained on a 32-core Intel Xeon CPU E5-2650 v2 with 2.60GHz and 32GB of memory, equipped with a GeForce GTX 970 graphics card with 4GB of memory.

4.6.2 Datasets and Parameters

Our test datasets include the Bonsai (256^3, 16 MB), the Hazelnut (512^3, 128 MB), the Flower (1024^3, 1 GB), and the Garlic (2048^3, 8 GB). All volumes are 8-bit micro-computed tomography (μCT) using X-rays. The Hazelnut, Flower and Garlic are available from [vmm, a], while the Bonsai is from [iap,]. For interactive visualization we apply a constant logarithmic 9-bit quantization scheme on the octree cores as proposed in [Suter et al., 2013], resulting in total tensor octree sizes of 3MB, 21.5MB, 167.7MB and 1.30GB, respectively. While this scheme results in lower compression quality than the adaptive quantization introduced in Ch. 3, its reconstruction is much more GPU-friendly. We used borders of size

$m = 8$ ($m = 10$ for the Garlic). Note that the Garlic in its raw form exceeds the available GPU memory by a factor of 2. It would be unfeasible to store the whole uncompressed volume in the GPU at all times; such large volumes motivate using compressed visualization techniques. We will use the shortcuts *FD* (filter before downsampling), *DF* (downsample before filtering), and *TAF* (tensor approximation domain filtering) as introduced earlier.

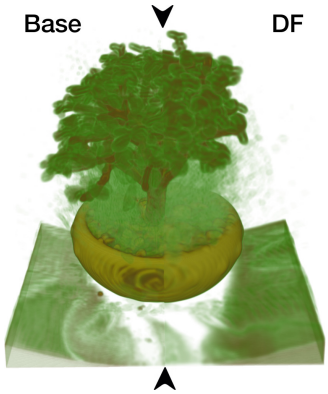
4.6.3 Multiresolution Remarks

In a multiresolution rendering where regions of different LOD resolutions are adjacent to each other, filtering after downsampling causes sharp discontinuities along the borders between LODs. As shown in the top row of Fig. 4.9, lower resolution spatial voxel filtering leads to sudden recognizable blurring and loss of high-frequency detail: in the right half-images, the edges are detected at a coarser scale than in the full-resolution version (left half-images). On the other hand, our method in the bottom row achieves a much more homogeneous overall result even across LOD resolution boundaries.

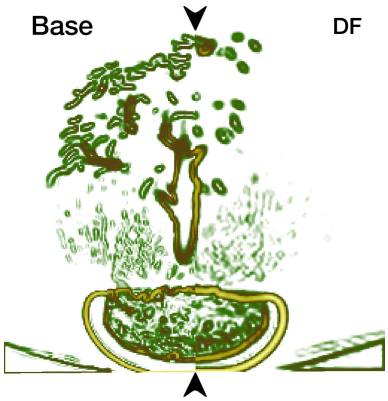
The Sobel operator from these experiments has a small kernel size of $K = 3$. A more correct application of this filter over lower-resolution bricks would require downsampling its kernel, but its size $3 \times 3 \times 3$ is too small. With the Tucker-octree filtering, however, this is feasible via global factor matrices, which have the full original resolution and can ensure edge detection at the correct spatial scale. These matrices have only size $I \times R$. A naive filter before downsampling (FD), on the other hand, would have to process the full reconstructed volume of size I^3.

4.6.4 Guided Filter Extension

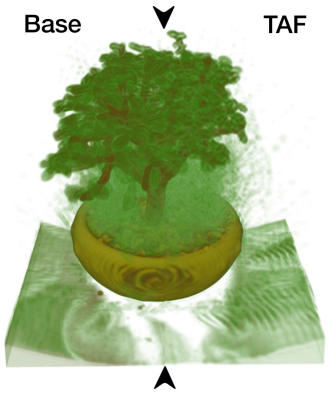
We have already shown how to compute the Sobel operator, which is non-linear but is nonetheless tractable via a combination of linear components. We have likewise implemented a tensor-reconstructed version of the guided filter [He et al., 2010], a non-linear edge-preserving filter popular for denoising that exhibits very good behavior near edges. Our implementation corresponds to the case where the guiding volume is identical to the input volume. A modification of our system was needed to this end: since the filter requires variances of all fixed size windows, we have to store a compressed version of the squared original data set. We only keep one set of factor matrices, as we have observed that they can be used to compress the squared bricks at a similar approximation error. Whenever the filter is applied, the squared brick cores are loaded from disk and kept synchronized with the regular brick cores. See Alg. 4 for the full details of our per-brick guided filtering plus reconstruction process. Note that, like the Sobel operator, we must perform the non-linear operations (line 6 to 8 in Alg. 4) after tensor reconstruction. In ad-



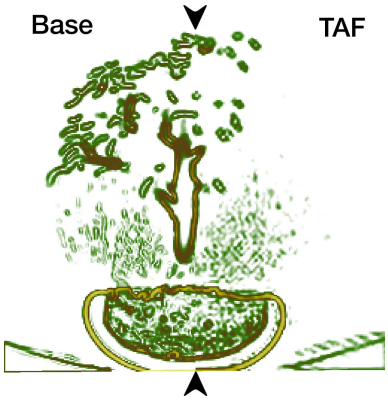
(a) baseline vs. downsample followed by filter (DF)



(b) baseline vs. downsample followed by filter (DF) (slice)



(c) baseline vs. ours (TAF)



(d) baseline vs. ours (TAF) (slice)

Figure 4.9: *Multiresolution filtering with the Sobel operator of the Bonsai: full spatial resolution* $I = 256$ *in the left image half; and* 1:2 *reduced resolution in the right half of the image. (a,b) Filtering in the spatial voxel domain after downsampling (DF) shown as volume and image slice views, with features suddenly becoming coarser on the right compared to the baseline on the left. (c,d) Reconstruction of filters applied to the basis factor matrices in the tensor domain (TAF) leads to less resolution dependent results.*

dition, the last box filters must be (approximately) adapted to the target resolution level. The error introduced this way is much smaller than with the naive approach as shown in Tab. 4.4.

Tabs. 4.2 to 4.4 show the relative numerical errors $\epsilon = \|\mathcal{T}_* - \widetilde{\mathcal{T}}_*\|/\|\mathcal{T}_*\|$ of filter results between the ground truth (FD) and DF as well as our method TAF for three different filters. The data set is tensor-compressed and reconstructed in all cases, the only difference being the downsampling stage (done after reconstruction

Algorithm 4 Apply guided filter and reconstruct one brick (i, j, k) at hierarchy level l with given window size w and damping factor δ

Require: Cores $\mathcal{B}'_{ijk}$ and $\mathcal{B}'^{*}_{ijk}$
Require: Corresponding factor chunks $\mathbf{U}'^{\cdots(n)}$ for $n = 1, 2, 3$
$\mathbf{B}^{(1)} := \text{downsample}(\text{boxfilter1D}(\mathbf{U}'^{\cdots(1)}, w))$
$\mathbf{B}^{(2)} := \text{downsample}(\text{boxfilter1D}(\mathbf{U}'^{\cdots(2)}, w))$
$\mathbf{B}^{(3)} := \text{downsample}(\text{boxfilter1D}(\mathbf{U}'^{\cdots(3)}, w))$
$\mathcal{M} := \mathcal{B}'_{ijk} \times_1 \mathbf{B}^{(1)} \times_1 \mathbf{B}^{(2)} \times_1 \mathbf{B}^{(3)}$ {Per-window mean}
$\mathcal{S} := \mathcal{B}'^{*}_{ijk} \times_1 \mathbf{B}^{(1)} \times_1 \mathbf{B}^{(2)} \times_1 \mathbf{B}^{(3)}$ {Per-window raw second moment}
$\mathcal{V} := \mathcal{S} - \mathcal{M}^2$ {Per-window variance}
$\mathcal{A} := \mathcal{V}/(\mathcal{V} + \delta)$ {Per-window a_k from [He et al., 2010]}
$\mathcal{B} := (1 - \mathcal{A})\mathcal{M}$ {Per-window b_k from [He et al., 2010]}
$\bar{w} := \max(1, \text{round}(w/(2^l)))$
$\bar{\mathcal{A}} := \text{boxfilter3D}(\mathcal{A}, \bar{w} \times \bar{w} \times \bar{w})$
$\bar{\mathcal{B}} := \text{boxfilter3D}(\mathcal{B}, \bar{w} \times \bar{w} \times \bar{w})$
$^{\downarrow l}\mathbf{U}'^{\cdots(1)} := \text{downsample}(\mathbf{U}'^{\cdots(1)})$
$^{\downarrow l}\mathbf{U}'^{\cdots(2)} := \text{downsample}(\mathbf{U}'^{\cdots(2)})$
$^{\downarrow l}\mathbf{U}'^{\cdots(3)} := \text{downsample}(\mathbf{U}'^{\cdots(3)})$
return $\bar{\mathcal{A}} \circ (\mathcal{B}_{ijk} \times_1 {}^{\downarrow l}\mathbf{U}'^{\cdots(1)} \times_2 {}^{\downarrow l}\mathbf{U}'^{\cdots(2)} \times_3 {}^{\downarrow l}\mathbf{U}'^{\cdots(3)}) + \bar{\mathcal{B}}$

but before spatial filtering in DF, and before reconstruction but after basis matrix filtering in TAF). The accuracy of the TAF processing is significantly higher than that of the spatial domain filtering applied after downsampling DF. The number of ranks was set to $R = I/2$ as no octree hierarchy was used for this error evaluation study.

4.6.5 Filtering Performance

To demonstrate the feasibility of our compression domain volume filtering approach, we measured the filtering and reconstruction performance of our proposed TAF compared to spatial domain filtering after downsampling (DF). In our test system, the rendering is temporarily halted whenever filtering is invoked, and resumed again after it is completed and all visible bricks have been reconstructed. This approach displays the full final filtered result as soon as possible, resulting in interactive response times (which depend on the filter complexity).

We rendered 4 datasets at an image resolution of 1024^2 pixels, and applied a DOG filter to them using the standard deviations of $\sigma_1 = 1$ and $\sigma_2 = 5$. The DOG has rank $S = 2$, as it can be written as a small CP decomposition with $(\lambda_1, \lambda_2) = (1, -1)$ and factor columns corresponding to two 1D Gaussian kernels.

Table 4.2: *Difference of Gaussians with $\sigma_1 = 1, \sigma_2 = 5$: Relative errors $\epsilon = \|\mathcal{T}_* - \widetilde{\mathcal{T}}_*\|/\|\mathcal{T}_*\|$ for different compressed datasets and downsampling factors: DF vs. the groundtruth (filtering, then downsampling) and TAF vs. the groundtruth. Our TAF approach achieves virtually no error at all resolution levels. Rank number was chosen as $R = I/2$*

ϵ		Downsampling factor			
		1	2	4	8
DF	**Bonsai**	0	0.8541	1.6948	3.2063
	Hazelnut	0	0.7871	1.5211	2.7220
	Flower	0	0.8133	1.7143	3.1692
	Garlic	0	0.7981	1.7209	4.4781
TAF	**Bonsai**	$5.15 \cdot 10^{-6}$	$2.89 \cdot 10^{-6}$	$2.66 \cdot 10^{-6}$	$5.59 \cdot 10^{-6}$
	Hazelnut	$4.54 \cdot 10^{-7}$	$2.91 \cdot 10^{-6}$	$3.09 \cdot 10^{-6}$	$4.55 \cdot 10^{-6}$
	Flower	$6.62 \cdot 10^{-7}$	$4.86 \cdot 10^{-6}$	$5.41 \cdot 10^{-6}$	$8.87 \cdot 10^{-6}$
	Garlic	$9.88 \cdot 10^{-6}$	$2.59 \cdot 10^{-5}$	$3.35 \cdot 10^{-5}$	$5.98 \cdot 10^{-5}$

The average timings for filtering and reconstruction as performed on the GPU are reported in Tab. 4.5, using three different filter sizes of $K = 5, 9, 17$. We used brick border overlaps of size 8. Note that the DOG standard deviations of $\sigma_{1,2}$ do not affect the timings which only depend on the filter size K. We measured and averaged 30 test runs for each configuration, and in order to ensure high quality rendering for realistic view configurations, the LOD parameter for voxel (brick) selection was chosen such that a voxel is projected onto no more than 2×2 screen pixels.

Even though the inaccurate DF approach does not require immediate reconstruction (as spatial filtering is done on the raw bricks), it still requires 3 separable filter passes over the N voxel bricks of size B^3 for each of the S ranks of the filter kernel. In contrast, the TAF approach performs the rank-S approximated DOG filter linearly on the R columns of the factor matrices only, independent of the number N and size B of the voxel bricks. This leads to extremely fast filtering in the compressed domain for virtually any filter size K, and also faster overall final results even after taking reconstruction into account as demonstrated in Tab. 4.5. Tensor reconstruction costs dominate the factor matrix filtering by several orders of magnitude, but are still consistently lower than traditional spatial filtering (DF). In other words, not only does tensor basis filtering offer more accurate results at lower resolution scales, but it is also faster even when accounting for the necessary brick reconstruction time.

For large volumes, accurate spatial filtering of the full resolution volume data followed by downsampling (FD) would be infeasible in real time. First, the dataset is limited in size since its full resolution version must fit into the GPU memory

Table 4.3: *Numerical relative errors $\epsilon = \|\mathcal{T}_* - \widetilde{\mathcal{T}}_*\|/\|\mathcal{T}_*\|$ for the Sobel operator with $R = I/2$. The error in our method is due to the non-linear operations performed after reconstruction, but is overall significantly smaller than the naive approach*

ϵ		Downsampling factor			
		1	2	4	8
DF	**Bonsai**	0	0.6030	1.3385	1.9533
	Hazelnut	0	0.5655	1.1121	1.4488
	Flower	0	0.5504	1.0976	1.5396
	Garlic	0	0.4768	0.8157	1.1637
TAF	**Bonsai**	$4.28 \cdot 10^{-7}$	0.1166	0.3053	0.5393
	Hazelnut	$5.56 \cdot 10^{-7}$	0.1257	0.3272	0.5532
	Flower	$8.88 \cdot 10^{-7}$	0.1573	0.3898	0.6026
	Garlic	$1.23 \cdot 10^{-6}$	0.2659	0.6160	0.8216

Table 4.4: *Numerical relative errors ϵ for the guided filter with $R = I/2, \delta = 5000$. Note the error incurred by non-linear operations, which is again smaller for TAF*

ϵ		Downsampling factor			
		1	2	4	8
DF	**Bonsai**	0	0.0740	0.1642	0.2439
	Hazelnut	0	0.1284	0.2753	0.3821
	Flower	0	0.1134	0.2603	0.3717
	Garlic	0	0.0351	0.0813	0.1285
TAF	**Bonsai**	$9.55 \cdot 10^{-8}$	0.0087	0.0231	0.0331
	Hazelnut	$1.10 \cdot 10^{-7}$	0.0142	0.0379	0.0608
	Flower	$1.46 \cdot 10^{-7}$	0.0115	0.0312	0.0487
	Garlic	$1.88 \cdot 10^{-7}$	0.0038	0.0099	0.0148

whenever the whole volume is visible. Second, the filtering operation would always entail maximal cost and be especially inefficient for strongly downsampled renderings of zoomed-out views on screen. Having to reconstruct the full resolution in all cases for filtering means that the LOD hierarchy is not exploited at all, which defeats the purpose of using multiresolution volume rendering in the first place.

To put the performance results in perspective to previous work, in [Treib et al., 2012] the reported average timings for uploading and decompressing a 1024^3-sized scalar field are $1.3s$, without accounting for the subsequent full resolution filtering costs. Our framework, on the other hand, exploits a hierarchical structure to render volume regions at adaptively different resolution levels. The LOD selection coupled with tensor-based compression is able to deliver complete filtered

results at times well below half a second. However, performance cannot directly be compared as in [Treib et al., 2012] the volume data has to be fully loaded on the GPU and is only processed after decompression, thus it does not support LOD based compression domain filtering and rendering.

Tabs. 4.6 and 4.7 show equivalent experiments for the Sobel operator and guided filter. The former has a fixed kernel size of 3, while for the latter we ran again 5, 9 and 17. The filtering timings in these cases combine both the convolution in the compressed domain and the necessary post-processing steps thereafter. While these filters are non-linear and more expensive than the DOG counterpart, they are still faster and more accurate than the naive DF implementation.

Table 4.5: *Difference of Gaussians: filtering and reconstruction times (in ms) for all four datasets, comparing DF vs. TAF for different kernel sizes. Timing values are averaged over 30 test runs using the DOG filter (with $\sigma_1 = 1$ and $\sigma_2 = 5$) of size $K = 5, 9, 17$. No immediate reconstruction is required for the spatial domain filtering DF.*

			Filtering (ms)			
		N. of bricks	Kernel size			Recons-truction (ms)
			5	9	17	
DF	**Bonsai**	55	114.93	154.56	234.77	–
	Hazelnut	111	224.46	297.59	445.12	–
	Flower	141	257.13	320.38	450.07	–
	Garlic	315	491.19	603.53	775.51	–
TAF	**Bonsai**	55	0.13	0.16	0.16	35.85
	Hazelnut	111	0.18	0.20	0.24	72.33
	Flower	141	0.29	0.32	0.42	91.73
	Garlic	315	0.50	0.61	0.87	204.24

Table 4.6: *Sobel operator: filtering and reconstruction times (in ms) for all four datasets*

		N. of bricks	Filtering (ms)	Reconstruction (ms)
DF	**Bonsai**	55	143.53	–
	Hazelnut	111	290.86	–
	Flower	141	369.56	–
	Garlic	315	825.31	–
TAF	**Bonsai**	55	44.84	48.25
	Hazelnut	111	90.44	97.68
	Flower	141	113.97	122.78
	Garlic	315	255.71	272.17

Fig. 4.10 shows exemplary screenshots of the renderer's state before and after the tensor-domain filtering process of applying the DOG and the guided filter alternatively.

Table 4.7: *Guided filter: filtering and reconstruction times (in ms) for all four datasets ($\delta = 3000$)*

			Filtering (ms)			
		N. of bricks	Window size			Recons-truction (ms)
			5	9	17	
DF	**Bonsai**	50	240.64	316.76	476.01	–
	Hazelnut	86	401.21	519.61	772.58	–
	Flower	247	1066.04	1320.99	1853.28	–
	Garlic	362	1325.84	1603.21	2020.86	–
TAF	**Bonsai**	50	159.23	196.84	270.87	42.42
	Hazelnut	86	265.49	323.09	441.08	72.87
	Flower	247	728.59	853.54	1105.52	207.72
	Garlic	362	955.71	1087.20	1290.67	306.55

Figure 4.10: *Top row: 4 volumes. Middle row: guided filter result ($w = 5$ and $\delta = 5$ for the Bonsai, $w = 5$ and $\delta = 10$ for the Hazelnut, $w = 9$ and $\delta = 10$ for the Flower, and $w = 17$ and $\delta = 30$ for the Garlic). Bottom row: Difference of Gaussians ($\sigma_1 = 1, \sigma_2 = 5$; filter kernel size $K = 17$ except for the Bonsai, which used $K = 9$).*

4.6.6 Rendering Performance

We also evaluated the interactive performance of our system with the three largest volumes, including the 8GB Garlic. We use a camera path starting from a zoomed-out view and progressing towards a more detailed close-up, with a filtering step applied at frame 437. The timings in Fig. 4.11 to Fig. 4.12 reveal that for interactive visual exploration the vast majority of time per frame is spent on rendering (volume ray casting), while the time required for incremental reconstruction of updated LOD volume bricks is much less significant. This behavior is consistent with the related TAMRESH system [Suter et al., 2013], as outlined in Sec. 4.5.5. Note that the total frame time is less than the sum of rendering and reconstruction, since LOD volume brick updates are asynchronous and incremental. Figs. 4.11 and 4.12 show the frame-by-frame computing costs, broken down by individual tasks (rendering, tensor reconstruction, and filtering).

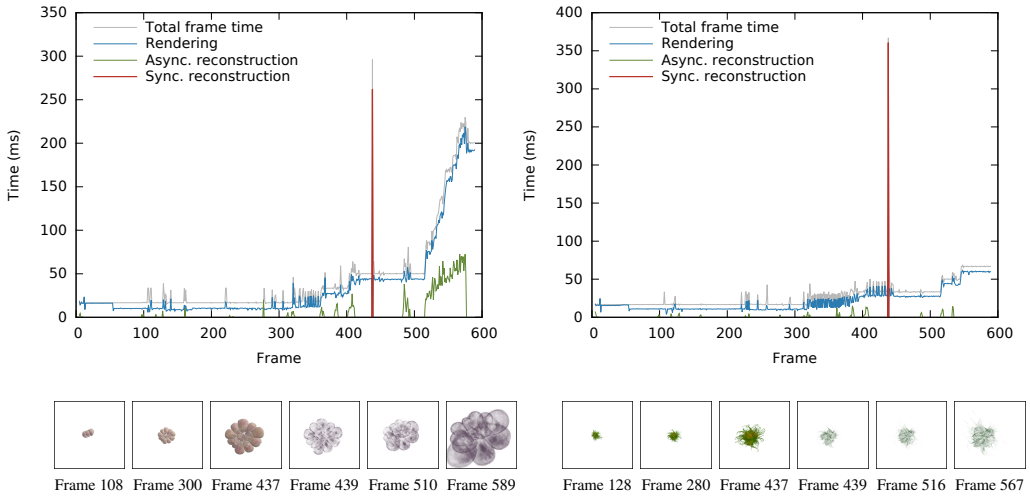


Figure 4.11: *Timings for the interactive visualization of the Garlic with a DOG filter operation (left) and the Hazelnut with the Sobel operator (right). We show the time required per frame for rendering (blue),* asynchronous *reconstruction for LOD updates (green) and* synchronized *reconstruction due to filtering (red), as well as the total time required to produce each frame (gray). The DOG uses kernel size* $K = 17$*, and* $\sigma_1 = 1, \sigma_2 = 5$.

A synchronized reconstruction of all visible volume bricks is needed after a filter operation has been invoked, as opposed to normal asynchronous LOD updates and rendering during viewpoint changes. Nevertheless, such reconstruction delays the next displayed frame only by a fraction of a second. In Figs. 4.11 to 4.12 this is indicated by the reconstruction cost peaks in red. Thus, after the user initiates a filter operation, the system can still react and update the rendered

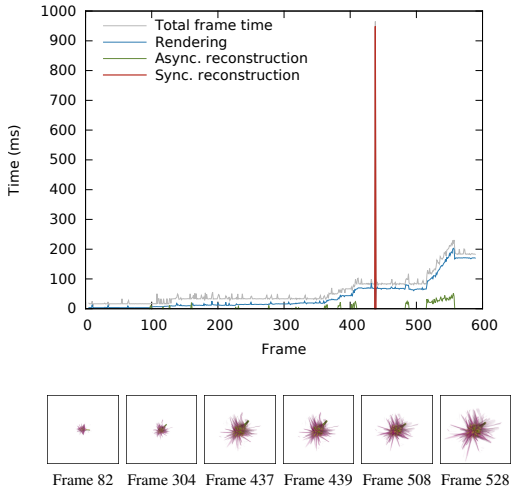


Figure 4.12: *Timings for the interactive visualization of the Flower with a guided filtering in-between ($w = 5, \delta = 1000$).*

image of the complete filtered result at an interactive rate. In the given example, a frame update time of about one third of a second is the only delay a user experiences for applying a (complex) filter operation, before interactive rendering of the now filtered volume resumes. Note that all overhead costs are reflected by the total frame time (gray), including initialization and finishing of a frame, CPU and GPU data management and transfer; and in the case of filtering, setting up and uploading filter kernels, etc.

4.7 Discussion

We presented a novel multiresolution volume filtering framework that can convolve variable-resolution data directly in the tensor compressed representation before reconstruction and rendering. The core idea is the fact that Tucker decomposition yields 1D basis components where the time-frequency duality can be inexpensively exploited. The proposed basis decomposition for each LOD hierarchy level actually contains the full resolution information, which ensures that a linear filter convolution can faithfully and homogeneously be applied at any spatial resolution and displayed at variable LOD. We show both numerical results and visual examples of our compression domain filtering approach, demonstrating its superior accuracy and higher speed compared to spatial domain filtering of downsampled volume data. This also applies to non-linear filters as long as they consist of linear components to some extent, as we have shown for the Sobel op-

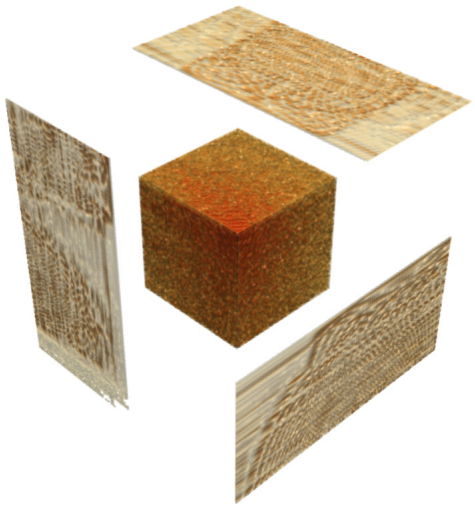
erator and the guided filter. The advantages become more significant with coarser LODs; this makes the proposed method especially suitable for large-scale volume representation, filtering and visualization. This is because larger models require more hierarchy levels and, potentially, present a larger disparity in resolution levels across the volume at each frame. Our method can compute and display filter operations at interactive rates on data that does not fit on GPU memory.

The proposed framework has certain limitations. First, non-linear operations for complex filters must be performed after tensor reconstruction, and thus such filters only benefit partially from our multiresolution system. On the other hand, linear but high-rank filters increase the computational cost. Lastly, the size m of the brick borders affects the filter correctness: small borders restrict the range of possible filter kernel sizes to $K \leq 2m$, while wide borders imply decreasing the compression quality.

In view of the very low costs of tensor-compressed domain volume filtering, and given that the Tucker reconstruction can be performed interactively, in the future we will investigate the feasibility of per-frame online reconstruction. Being able to render while decompressing on the fly would allow loading and storing the data always in its compressed format, which is a more space-efficient usage of the GPU memory. This would in turn relax the need for frequent out-of-core data fetching, further alleviating the upload overhead in the rendering pipeline.

C H A P T E R 5

HISTOGRAM RECONSTRUCTION



5.1 Overview

In the previous chapter we showed how to apply filtering operations to large, tensor-compressed volumes. However, filters and convolution are not the only analysis and feature extraction operations that we would like to exploit. Histograms, for example, are widely used in many visualization applications and are important tools in segmentation, object tracking and classification, volume rendering, and more. Recent advances in data acquisition technology are giving rise to increasingly large data sets that often require histogram queries over large data regions. Developing algorithms for compact representation and fast histogram computation is thus an area of active research. The integral histogram [Porikli, 2005], that we denote IH, is a very time-efficient way to obtain a histogram over any rectangular axis-aligned region in a Cartesian space. It extends the concept of summed area table (SAT) [Crow, 1984] by storing a cumulative histogram at each table entry. In exchange for its high query speed, this data structure is very redundant: it causes a manifold increase of the input data size and may often exceed available memory resources. For instance, in the present chapter we consider a 128MB micro-CT. If 64 histogram bins are used, the resulting 32GB integral histogram (4-byte integers are needed) does not fit into GPU or even main memory in most desktop environments, and queries require several expensive non-contiguous disk accesses. This makes compressing histograms in their integral form an important target. The goal is to reduce storage needs to a manageable amount while still allowing a faster histogram calculation than a brute-force traversal of the original data.

Contribution

We contribute the first tensor-based algorithm for compressed IH look-up. The proposed system (Fig. 5.1) compresses first the IH of the input image or volume along with the indicator functions for the desired types of region of interest (ROI): box, Gaussian, sphere, etc. For look-up the compressed IH is convolved with the query ROI and reconstructed to produce the desired histogram array. Our method a) allows decreasing both size and query time as needed, at the expense of a variable loss in response accuracy due to compression; and b) can handle arbitrary regions, as opposed to other methods that are limited to axis-aligned rectangles. Furthermore, while directly computing a histogram over a ROI requires many non-sequential memory accesses, in tensor reconstruction the compressed representation is always traversed in the same dimensional order, which allows a more memory-optimized computation.

Notation

ROI sizes are denoted with the letter K. When ROI indicator functions are compressed via tensor decompositions, we write their ranks as S. Last, B is the number of histogram bins.

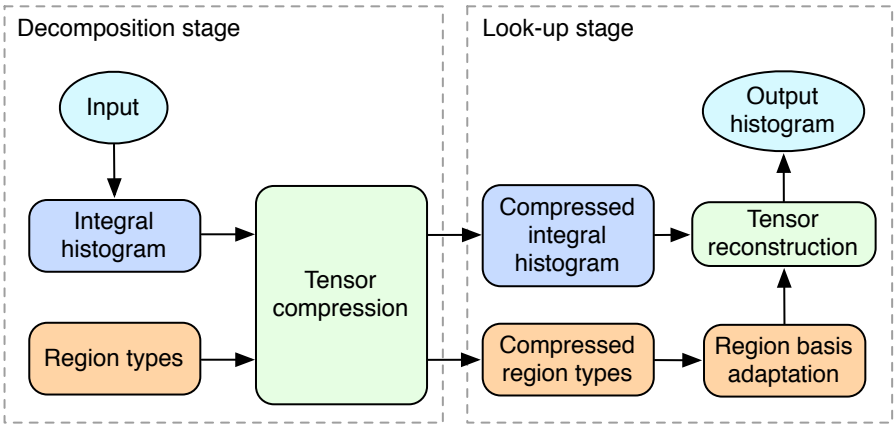


Figure 5.1: *Pipeline of tensor-based histogram decomposition and reconstruction.*

5.2 Background

5.2.1 Efficient Multidimensional Histograms

When processing large regions of interest within multidimensional data sets, computational costs are a usual concern in real-time visualization applications. A common approach for efficient histogram generation is avoiding redundant computations, e.g. with incremental sliding windows [Weiss, 2006; Wei and Tao, 2010] or identifying overlapping regions [Berger et al., 2012]. Alternatively, other methods focus on directly computing the desired histogram features from the data [Kass and Solomon, 2010; Hadwiger et al., 2012b]. In [Hadwiger et al., 2012b] and [Sicat et al., 2014], pixel or voxel neighborhood information is compactly represented as a sparse sum of Gaussian probability density functions, which allows for convenient retrieval of features used in filtering and for applying a transfer function in volume rendering. On the other hand, some tools compute histograms of a transformation of the data; some examples include the histogram of gradients (HoG, a method to extract robust feature descriptors) and its variants [Dalal and Triggs, 2005], and the 2D histogram, which takes into account gradient information (useful for e.g. segmentation and transfer function selection).

5.2.2 Summed Area Tables and Integral Histograms

The SAT is a data structure for fast and constant-time integral look-up over rectangular regions [Crow, 1984]. In 1D it is just a discretization of the Fundamental Theorem of Calculus: by storing the cumulative sum of an array $F[n] := \sum_{i=0}^{n-1} f[i]$ (with $F[0] := 0$), one can compute an arbitrary definite integral $\sum_{a}^{b-1} f(i)$ in constant time as the difference of F at the interval borders, $F[b] - F[a]$. For higher dimensions $N \geq 2$ the integral is given by the sum of values (with alternating signs) from the 2^N corners of the box region. SATs have been proven useful in computer graphics, especially in volume rendering, and parallel integration and filtering algorithms have been developed for the GPU for fast SAT generation [Hensley et al., 2005; Nehab et al., 2011; Schlegel et al., 2011].

The IH emerged as a natural extension of SATs for histogram look-up by adding a bin dimension [Porikli, 2005], and SAT querying algorithms can be in principle adapted to query it. The idea works as follows. Each bin $b = 1, \ldots, B$ is representable as a binary mask with the same shape as the original data: an entry is set to 1 if and only if the corresponding input pixel has the value b. Then, each binary mask's integral is encoded using an SAT, and all resulting tables are stacked along a new dimension of size B to yield the final IH. To retrieve a histogram from a certain target rectangle, an SAT query is performed for each bin over that region. Fig 5.2 illustrates the IH look-up for the 2D (image) case.

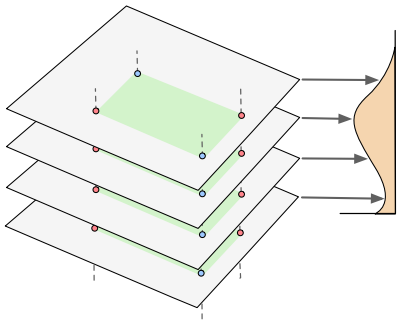


Figure 5.2: *Querying a 2D integral histogram over a rectangular region (highlighted in green) requires 4 look-ups per bin.*

The IH data structure generalizes easily to high dimensions [Tapia, 2011], and requires $B \cdot 2^N$ value look-ups to compute the final result. A slice of the size of the original data must be stored for every value bin, which can render the uncompressed IH too large to be used interactively. Compression approaches have been explored; for example, rectangle area sums can be efficiently computed over wavelet-decomposed data [Wu et al., 2000]. [Lee and Shen, 2013] proposed WaveletSAT, a lossless wavelet-based compression scheme for integral

histograms. They report fast query times and compression rates of 1:8 and more compared to the original IH size, for data up to 256^3 voxels. However, the method is still limited to axis-aligned rectangular regions, and can only deal with other types of regions by approximating them as sums of rectangles. An improvement in this direction was proposed by [Heckbert, 1986]. Based on repeated SAT integration, the author devised a procedure to obtain integrals across regions which are defined by polynomials and not just rectangles. Non-rectangular regions are often preferred indeed, e.g. Gaussian (to achieve rotation-invariance) or 3D cross-neighborhoods [Lundstrom et al., 2006], or complex segmented target regions. In this context, our tensor-compressed histogram querying algorithm emphasizes low storage needs and flexibility regarding target region shape.

5.3 Integral Histogram Tensor Compression

Let $\mathcal{T}$ be a multidimensional array of size $I_1 \times \cdots \times I_N$, for example an image ($N = 2$) or a volume ($N = 3$). The full IH $\mathcal{I}$ of $\mathcal{T}$ can be computed from its level-set stack $\mathcal{L}$, which has size $I_1 \times ... \times I_N \times B$ and is defined as:

$$\mathcal{L}[x_1, ..., x_N, b] = \begin{cases} 0 & \text{if } \mathcal{T}[x_1, ..., x_N] \neq b \\ 1 & \text{if } \mathcal{T}[x_1, ..., x_N] = b \end{cases}$$

To eventually get $\mathcal{I}$, one needs to calculate all *cumulative* partial sums of $\mathcal{L}$'s slices along the last dimension. The result has size $(I_1 + 1) \times \cdots \times (I_N + 1) \times B$. Element-wise, each $\mathcal{I}[x_1, ..., x_N, b]$ is defined as:

$$\begin{cases} 0 & \text{if } i_n = 0 \text{ for some } n \\ \sum_{i_1,...,i_N=1}^{x_1,...,x_N} \mathcal{L}[i_1 - 1, ..., i_N - 1, b] & \text{otherwise} \end{cases}$$

Note that whereas $\mathcal{L}$ is highly sparse, the cumulative $\mathcal{I}$ is not. While one can already use $\mathcal{L}$ to compute histograms by direct integration, we choose to compress the array $\mathcal{I}$ instead. The motivation is that we found that histograms reconstructed from a tensor-compressed $\mathcal{I}$ are much more accurate for the same number of coefficients. The columns $\mathcal{L}[x_1, ..., x_N, :]$ have little correlation between each other: any value change between two points (e.g. $\mathcal{T}[x_1, ..., x_N] \neq \mathcal{T}[y_1, ..., y_N]$) implies that $\langle \mathcal{L}[x_1, ..., x_N, :], \mathcal{L}[y_1, ..., y_N, :] \rangle = 0$. On the other hand, every column $\mathcal{I}[x_1, ..., x_N, :]$ depends on the whole region between it and the origin $\mathcal{I}[0, \ldots, 0, :]$; thus, columns close to each other have very similar contents. To support this observation, we show in Fig. 5.3 the histogram reconstruction accuracy from compressing $\mathcal{L}$ versus compressing $\mathcal{I}$.

Now, the question of how to efficiently compress $\mathcal{I}$ arises. The most straightforward approach would be to build the full $\mathcal{I}$ explicitly, and then compress it.

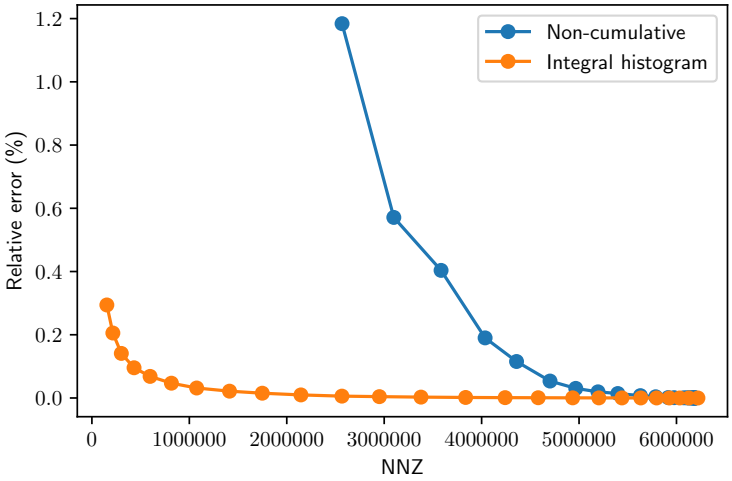


Figure 5.3: *Average histogram accuracy for rectangular ROIs of size* 32^3 *within a brick of size* 64^3 *(Bonsai data set). The target error during compression* ϵ *is decreased as we move left to right; each result dot is the average reconstruction error of 100 regions placed at random. Using the IH* $\mathcal{I}$ *instead of the level-set* $\mathcal{L}$ *leads to a much smoother degradation as well as faster histogram reconstruction over rectangular regions; see also Sec. 5.4.*

However, note that for medium or large models this quickly becomes impractical. Fortunately, thanks to the versatility of the TT format, we are able to split up any tensor and incrementally build up its compression. We namely combine two steps:

- An algorithm to compress any *slice* of the IH, i.e. on a bin-by-bin basis;
- A *sum-and-compress* procedure to combine and merge existing compressed slices into one single compressed tensor.

We detail next how we address each of these separate steps.

5.3.1 Slice Compression

Any full dense tensor can be decomposed in the TT format via the TT-SVD algorithm [Oseledets, 2011], which successively unfolds the data and computes the singular value decomposition (SVD) on the result, one mode at a time. The desired target error $0 \leq \epsilon$ is defined beforehand and is the only parameter. TT-SVD guarantees that the final relative error will be no larger than the target ϵ.

Let $\mathcal{S}_b$ be the b-th bin slice of the IH, i.e. $\mathcal{I}[:, \ldots, :, b]$. We compute it independently of all other bin slices: first we create a binary mask from the input data, and then we compute its SAT. We now apply the TT-SVD algorithm to compress $\mathcal{S}_b$ into a TT tensor with N cores at a prescribed accuracy ϵ. Note that this

ϵ affects only the global quality of the IH and that relative errors of individually reconstructed histograms (Secs. 5.4 and 5.5) will differ, especially with respect to the region size.

The original TT-SVD uses a sequence of tensor unfoldings, interleaved with SVD decompositions that progressively remove one dimension at a time to produce one core and thus reduce the overall tensor size [Oseledets, 2011]. Such unfoldings flatten the tensor into a matrix in various orderings so as to remove redundancy along one dimension at a time. We observed that in our applications these unfoldings are often very tall or wide matrices. Therefore, and similarly to our compression algorithm in Ch. 3, we obtain the necessary singular vectors via the eigenvalue decomposition of covariance matrices. We call this modified routine TT-EIG and found it to be significantly faster than standard SVD. Although these two approaches have a different numerical behavior and their resulting singular vectors and values may differ, these differences are negligible for the compression rates and real-world multidimensional signals that we tested. The complete procedure is given in the Appendix.

5.3.2 Sum-and-Compress

As argued before, IHs are often too large to be manipulated as a whole. Given a multidimensional data set $\mathcal{T}$ and a number of bins B, we propose an incremental algorithm that progressively adds one slice at a time. We start with $b = 0$, and at each slice $0 \leq b < B$ we take a compressed IH of bins $[0, b-1]$ and produce an IH for bins $[0, b]$:

1. We compute the b-th slice of the IH (of size $I_1 \times \cdots \times I_N$) and compress it into a TT tensor $\mathcal{A}$ as outlined above in Sec. 5.3.1.

2. We add an *indexing core* at the end with B elements that encodes the value b using *one-hot encoding*, i.e. all its elements are 0, except the b-th one which is 1. As a result, the modified TT $\mathcal{A}$ encodes now a tensor of $N+1$ dimensions with size $I_1 \times \ldots I_N \times B$. It has zeros everywhere outside its b-th slice.

3. We add this TT to our partial IH. Adding two tensors is a straightforward operation [Oseledets, 2011]. The result encodes now an IH for all bins $[0, b]$.

4. We recompress the result to remove any redundant information that may have appeared. We achieve this by means of the *TT-round* procedure, which is related to TT-SVD, its full details are given in [Oseledets, 2011].

Our incremental algorithm Alg. 5 given below exploits the fact that each compressed slice is small enough to be handled separately. Although the ranks are

reduced after each recompression, the overall IH compressed size tends to grow as we work our way towards the last bin. Steps 3 and 4 are computationally intensive, especially when the accumulated IH is large. In order to reduce this cost we merge the IH slices in a recursive, binary-tree fashion: slice 0 is merged with slice 1, the result is merged with the combination of 2 with 3, and so on. Each tree root $\mathcal{R}_c$ encodes the IH for 2^c consecutive slices, and we create $\mathcal{R}_{c+1}$ by joining it with the next 2^c slices. This strategy ensures that, asymptotically, most merge operations involve only small tensors.

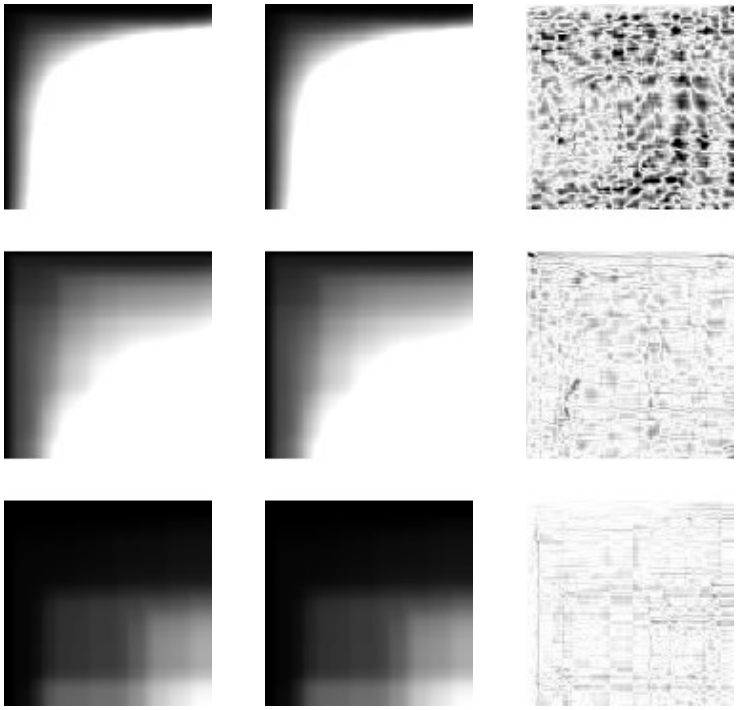
Fig. 5.4 shows the accuracy of the proposed IH compression over a 4096^2 grayscale image.

Algorithm 5 Build an integral histogram $\mathcal{I}$ from an input tensor $\mathcal{T}$ with B bins and error parameter ϵ, using sum-and-compress combined with TT-EIG.

for $b = 0, \dots B-1$ **do**
  // Compute and compress the b-th slice of the IH of $\mathcal{T}$
  $\mathcal{A} :=$ TT-EIG$(\mathcal{I}[:, \dots, :, b], \epsilon)$
  $\mathcal{B} :=$ ones$(1 \times B \times 1)$ *// This core indexes the bin*
  $\mathcal{B}[0, b, 0] := 1$ *// One-hot encoding*
  $\mathcal{A} := [[\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(N)}, \mathcal{B}]]$ *// Append the indexing core*
  $c := 0$
  while $\mathcal{R}_c$ exists **do**
    $\mathcal{A} :=$ TT-ROUND$(\mathcal{A} + \mathcal{R}_c, \epsilon)$
    Delete $\mathcal{R}_c$
    $c := c + 1$
  end while
  $\mathcal{R}_c := \mathcal{A}$
end for
$\mathcal{I} =$ zeros$(I_1 \times \cdots \times I_N \times B)$
for $\mathcal{R}_c$ **do**
  // Gather and sum all remaining roots
  $\mathcal{I} := \mathcal{I} + \mathcal{R}_c$
  Delete $\mathcal{R}_c$
end for
return $\mathcal{I}$

5.4 Histogram Reconstruction

All tensor decompositions are multilinear in nature. This is a key feature that we exploit in this section and makes several compressed-domain operations very efficient.



(a) Original (b) After compression (c) Error (x100 amplified)

Figure 5.4: *Integral histograms are well compressible using the TT decomposition model. Column (a): three evenly-spaced IH bins for the Waterfall image with $B = 64$. Column (b): IH bins after TT compression to $(1, 64, 51, 1)$ ranks. Column (c): absolute difference, magnified 100-fold to ease visual appreciation.*

5.4.1 Spatial Tensor Basis Manipulation

Linear operations can be applied on a compressed IH $\mathcal{I} = [[\mathcal{I}^{(1)}, \ldots, \mathcal{I}^{(N)}, \mathcal{B}]]$ by slice-wise manipulating its TT cores. If instead of a core $\mathcal{I}^{(n)}$ we use a weighted sum of its slices $\mathbf{I}^{(n)} := \sum_{i=0}^{I_n} \mathbf{c}[i] \cdot \mathbf{I}^{(n)}[:, i, :]$ (where $\mathbf{c}$ is a vector with $I_n + 1$ entries), then the reconstruction produces a linear combination of the corresponding hyperslices along the n-th mode. In tensor notation, this means that

$$[[\mathcal{I}^{(1)}, \ldots, \mathcal{I}^{(n)} \times_2 \mathbf{c}, \ldots, \mathcal{I}^{(N)}, \mathcal{B}]] = \mathcal{I} \times_n \mathbf{c}, \tag{5.1}$$

where we have used tensor-times-vector (TTV) products. An n-mode TTV computes the dot product between all tensor vectors along its n-th dimension and an external vector.

Eq. 5.1 allows us to reconstruct histograms from a compressed IH over various

ROI types: next we cover the case of rectangular regions, and later in Sec. 5.5 we extend the method to non-rectangular query ROI shapes.

5.4.2 Querying a TT-Compressed IH

Let $\mathcal{T}$ be a tensor data set of size $I_1 \times \cdots \times I_N$, and ω a target region of interest (ROI), defined by the following indicator function,

$$\omega[x_1, \ldots, x_N] = \begin{cases} 1 & \text{if } x_1 \in [i_1, j_1), \ldots, x_N \in [i_N, j_N) \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathcal{S}$ be the summed area table of $\mathcal{T}$, and 0 and 1 denote either i or j for the 2^N vertices of ω. Thus, $\mathcal{S}[0, ..., 0]$ corresponds to $\mathcal{S}[i_1, ..., i_N]$, $\mathcal{S}[1, 0, ..., 0]$ to $\mathcal{S}[j_1, i_2, ..., i_N]$, and so on. The summation of $\mathcal{T}$ over ω can be obtained from $\mathcal{S}$ by adding and subtracting the following 2^N terms [Tapia, 2011]:

$$\sum_{x_1=i_1}^{j_1-1} \cdots \sum_{x_N=i_N}^{j_N-1} \mathcal{T}[\mathbf{x}] = \sum_{p \in \{0,1\}^N} (-1)^{N-\|p\|_1} \cdot \mathcal{S}[p] \tag{5.2}$$

The norm $\|\cdot\|_1$ counts the number of elements of value 1 in $[p]$ and determines the parity of each summand. For example, the SAT result in the 2D case is $\mathcal{S}[j_1, j_2] - \mathcal{S}[j_1, i_2] - \mathcal{S}[i_1, j_2] + \mathcal{S}[i_1, i_2]$. For N dimensions this requires 2^N look-ups in the multiarray $\mathcal{S}$, with 2^{N-1} positive terms and 2^{N-1} negative terms.

Thanks to multilinearity, tensor decompositions allow conversion of multidimensional operations into a sequence of equivalent 1D operations. In particular we can easily translate Eq. 5.2 into the TT compressed domain: it suffices to subtract the core slices that delimit the rectangle borders. Thus, we define

$$\mathbf{I}^{(n)} := \mathcal{I}^{(n)}[:, j_n, :] - \mathcal{I}^{(n)}[:, i_n, :]$$

for $n = 1, \ldots, N$; each is a matrix of shape $R_{n-1} \times R_n$. The last core $\mathcal{B}$ is unchanged: it is not a spatial axis, since it encodes the histogram bins. The desired histogram is then obtained as

$$\mathbf{I}^{(1)} \cdots \mathbf{I}^{(N)} \cdot \mathcal{B}[:, :, 0]$$

See the Appendix for an example illustration of these reconstruction formulas in the 2D case.

5.5 Non-rectangular ROIs

So far we have outlined how to integrate histograms only over rectangular regions. However, more general ROIs are supported as well, i.e. where the membership of every point to the query region is weighted by a real value in $[0, 1]$. Such cases are interesting e.g. when rotation-invariant descriptors are desired (i.e. where the region's membership function is radial) or when the membership is only estimated with a certain probability below 1 (e.g. a confidence heatmap arising from a segmentation procedure). Note that an IH in its original form is only optimal for rectangular regions and is not easily applicable in these more general cases.

In this section we make a distinction between reconstructing single histograms (as before) and the more general case in which blocks of histograms are desired.

5.5.1 Non-rectangular Reconstruction

In order to reconstruct one histogram from $\mathcal{I}$ over non-rectangular region we use the following identity:

$$\int_0^K f(x) \cdot g(x)\, dx = -\int_0^K \left(\int_0^x f(y)\, dy \right) \cdot g'(x)\, dx$$

which holds if either $f(x)$ or $g(x)$ are 0 at both $x = 0$ and $x = K$. In its discrete form, it means that to find the sum of a weighted array we can just compute the sum of the cumulative array (the IH in our case), weighted by the derivative of the weights. Each IH slice is represented by f, while g plays the role of our ROI's indicator function. In practice, we need the TT decomposition of our target ROI. Let $\mathcal{R} = [[\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(N)}]]$ be an N-dimensional TT tensor encoding the ROI with ranks $S_1, \ldots, S_{N-1}$, and $\Delta\mathcal{R}$ its derivative along all dimensions. Each bin b of the desired histogram follows from a dot product between $\Delta\mathcal{R}$ and a window on the corresponding slice of $\mathcal{I}$:

$$\langle \mathcal{I}[i_1{:}j_1, \ldots, i_N{:}j_N, b], \Delta\mathcal{R} \rangle \tag{5.3}$$

We compute the tensor dot product from Eq. 5.3 by successively fusing the cores of $\Delta\mathcal{R}$ and $\mathcal{I}^{(n)}$ via tensor contractions (for more details on dot products in the TT format, see [Oseledets, 2011]). The last core simply indexes the bins and does not vary; in particular we compute the dot product for all bins at once.

In Table 5.1 we show the asymptotic costs in terms of space, precomputing time and query time for several histogram reconstruction methods, including ours and the related method WaveletSAT [Lee and Shen, 2013] (which is lossless and limited to rectangular ROIs only).

Algorithm 6 Given an IH $\mathcal{I}$ compressed with ranks $R_1, \ldots, R_N$, reconstruct a histogram over a non-rectangular region of bounding box $[i_1, j_1] \times \cdots \times [i_N, j_N]$ whose indicator function is approximated by a TT $\mathcal{R} = [[\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(N)}]]$ with ranks $S_1, \ldots, S_{N-1}$

// Compute $\Delta\mathcal{R}$ from $\mathcal{R}$ by deriving along all axes
for $n = 1, \ldots, N$ **do**
  $\Delta\mathcal{R}^{(n)}[:, 0, :] = \mathcal{R}^{(n)}[:, 0, :]$
  for $i = 1, \ldots, I_n - 1$ **do**
    $\Delta\mathcal{R}^{(n)}[:, i, :] := \mathcal{R}^{(n)}[:, i, :] - \mathcal{R}^{(n)}[:, i-1, :]$
  end for
end for
$\mathcal{F} := \big(1\big)$ *// 1×1 tensor*
for $n = 1, \ldots, N$ **do**
  $\mathcal{F} := \mathrm{contraction}(\mathcal{F}, \mathcal{I}^{(n)})$
  // $\mathcal{F}$ has now size $I_n \times R_n \times S_{n-1}$
  $\mathcal{F} := \mathrm{contraction}(\mathcal{F}, \Delta\mathcal{R}^{(n)})$
  // $\mathcal{F}$ has now size $R_n \times S_n$
end for
// $\mathcal{F}$ has now size $R_N \times S_N = R_N \times 1 = R_N$
return $\mathcal{F} \cdot \mathcal{B}[:, :, 0]$ *// Vector-matrix product: the result is a histogram with B elements as expected*

	Total space	Precomputing time	Query time (box ROI)	Query time (box ROI)
Brute force	I^N	0	$O(K^N)$	$O(K^N)$
IH	$I^N B$	$O(I^N B)$	$O(2^N B)$	$O(2^N K^N B)$
IH (precomputed convolution)	$I^N B$	$O(N I^N \log_2(I) B)$	$O(2^N B)$	$O(2^N B)$ (assumes ROI shape never changes)
WaveletSAT [Lee and Shen, 2013]	No closed formula	$O(I^N \log_2(N)^N)$	No closed formula	Not supported
TT (proposed method)	$(N-1)IR^2+$ $IR+RB$	$O(NIR^3)$ [Oseledets, 2011]	$O(NKR^2)+RB$	$O(NIR^2S^2)+O(RB)$

Table 5.1: *Asymptotic space and time costs for several methods. The whole uncompressed IH is used in the second and third algorithm. The third algorithm filters each bin slice by means of the fast Fourier transform (FFT), and the resulting convolutions are stored separately to achieve faster query times (but this assumes an invariant filter kernel).*

5.5.2 Histogram Field Reconstruction

In this last contribution section we extend our framework to reconstruct many histograms at once, namely over a collection of sliding ROIs that produce a *histogram field*. This operation gives a mapping between each input pixel/voxel and the histogram of its neighborhood, and is equivalent to a bin-by-bin convolution with the ROI's indicator function. To this end we now use the identity

$$f * g = \left(\int f \right) * g'$$

instead of the one from Eq. 5.3. In discrete form, it allows us to compute the histogram of the input over a sliding ROI by simply convolving our IH $\mathcal{I}$ with the region's derivative $\Delta\mathcal{R}$. In other words, we need to convolve two tensors in the TT format. We implement this for separable regions $\omega = \omega^{(1)} \otimes \cdots \otimes \omega^{(N)}$, a case that is handled by simply convolving the TT cores along the spatial dimension [Rakhuba and Oseledets, 2015]. More specifically, we build a new tensor $\mathcal{I}_* = [[\mathcal{I}_*^{(1)}, \ldots, \mathcal{I}_*^{(N)}, \mathcal{B}]]$ with each n-th core defined as

$$\mathcal{I}_*^{(n)} := \mathcal{I}^{(n)} * \Delta w^{(n)} \tag{5.4}$$

where $*$ denotes convolution of a 3D tensor (the TT core) with a vector along its second (spatial) axis and Δ is again the discrete differential operator, here applied to a vector.

5.6 Results

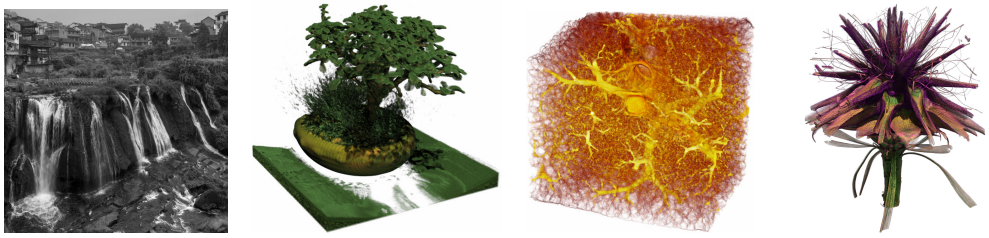
We implemented[1] and evaluated the proposed compression/decompression strategies on one image, three scalar volumes and one volume vector field.

5.6.1 Hardware and Software Used

Our code and tests were written in Python 3.5 and run on an Intel i7-4810MQ at 2.80GHz with 4 cores and 4GB of main memory. Operations for TT manipulation make use of the ttpy toolbox [ttp,], a Python/FORTRAN library based on an earlier MATLAB package for the TT format. All compressed tensors as well as reconstructed histograms are handled in 64-bit floating point format. We compare our approximate results with respect to groundtruth histograms computed via brute-force counting, namely NumPy's `histogram()` function. As we are aware that brute-force histograms are highly-parallelizable, we also implemented a GPU-accelerated version of single histogram reconstruction that takes advantage of CuPy [cup,], a high-level NumPy-like interface for CUDA that can also seamlessly integrate handwritten kernels. Our kernel exploits CUDA's `atomicAdd()` operation and CuPy's `bincount_kernel`, and was run using an NVIDIA Quadro K2100M GPU with 2GB of memory.

5.6.2 Scalar Field Integral Histograms

We have evaluated our proposed compression with four scalar fields including a 4096^2 grayscale picture of a *Waterfall* [wat,] and three μCT scans of a *Bonsai* tree [vol,] (size 256^3), a *Lung* (size 512^3) and a *Flower* [vmm, b] (size 1024^3), all shown in Fig. 5.5 and originally having 8-bit depth.



(a) Waterfall, 4096^2 (b) Bonsai, 256^3 (c) Lung, 512^3 (d) Flower, 1024^3

Figure 5.5: *The four scalar fields used in our experiments: an image (a) and three μCT volumes (b), (c), (d).*

[1] Our code is available at *https://github.com/rballester/tthistograms*.

As a preprocessing step, pixels or voxels are linearly mapped into the desired number of bins B: element-wise, $x \mapsto \lfloor x \cdot B/256 \rfloor$, with $B = 64$ and $B = 128$ in our experiments. Table 5.2 summarizes our compression algorithm's performance and results during the decomposition stage: time needed, compressed size, TT ranks, etc. Our compressed IH $\mathcal{I}$ takes at most a few hundred MB for the largest datasets, mainly depending on each data set's complexity; thus it is always one or two orders of magnitude smaller than the full uncompressed IH.

	Waterfall	**Bonsai**	**Lung**	**Flower**	**Hurricane**
Size (MB)	16	16	128	1024	21.70
B	64	128	128	128	128
Computing time (s)	1282.26	1085.48	2070.83	28138.90	4958.32
Full IH (GB)	4	8	64	512	10.84
Error target ϵ	0.0005	0.0001	0.0002	0.000015	0.0001
Compressed IH (MB)	181.32	38.34	205.15	85.61	281.28
IH compression ratio	22.59	213.69	319.45	765.50	39.49
TT ranks	100, 57	53, 137, 89	87, 271, 106	53, 121, 37	127, 535, 56

Table 5.2: *Compression results for the 5 data sets we have tested.*

Fig. 5.6 demonstrates the results for two separable ROI query examples on the Waterfall image and their resulting histograms (exact and approximated). The query regions are a square and a 2D Gaussian, and are queried as described in Sec. 5.4 and Sec. 5.5 respectively.

Using the four scalar field data sets as a benchmark, we have gathered query performance results for a range of regions. These results are reported in Fig. 5.7. For each data set we consider ROIs of increasing size placed on the data set's center. For Gaussian regions we chose σ as 1/4 of the region's size in all cases. Each histogram was reconstructed 3 times; the final time is the average.

In relation to the brute-force approach, our new method performs significantly faster, even compared to a CUDA implementation, with more pronounced benefits for larger queries and datasets. As we expected, slicing operations (besides the bin counting itself) are a significant bottleneck in brute-force methods. This is much less of a burden for TT reconstruction, since it works with whole core slices and therefore needs fewer random accesses. The relative query error due to the compression is also shown to drastically diminish with increasing ROI query sizes. Note that rectangular queries demand a constant number of operations with our method, and this is reflected in the roughly constant TT reconstruction time. This is not the case for the Gaussian, whose costs increase moderately. Further discussions are given below in Sec. 5.7.

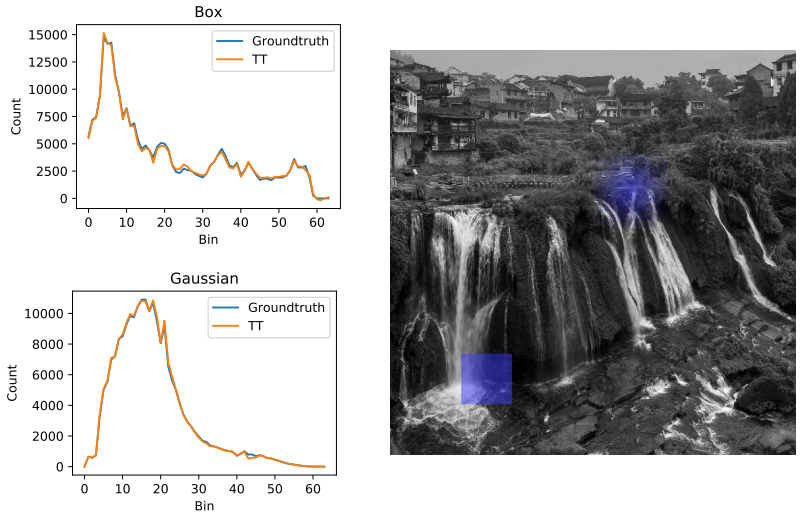


Figure 5.6: *Example: 64-bin histograms across two rank-1 rectangular 2D regions using TT compression to* $(64, 51)$ *ranks, with a 1:78.7 memory reduction from the original IH size. The TT box look-up over* 512^2 *pixels took 0.08ms while the brute-force took 4.2ms. The Gaussian query over* 768^2 *pixels took 4.4ms while the brute-force took 23.0ms.*

5.6.3 Vector Field Entropy

Our second experiment involves a 3D vector field, namely the wind speed coordinates from the 25th timestep of the Hurricane NCAR data set, available from [hur,]; see Fig. 5.8. The original vector field contains 100 height slices, each of which has 500×500 data points. We removed the first 9 height slices, since they intersect with parts of Florida's terrain and thus contain missing values. The experiment consisted in computing 3D Shannon entropy fields based on the wind orientation histogram; such fields have applications in feature extraction, information-aware streamline placement and visualization, etc. [Xu et al., 2010; Lee and Shen, 2013; Chen et al., 2016]. To this end we quantize the wind direction at each voxel using 128 quasi-uniform regions on the spherical surface. The local histogram for a voxel is thus defined as the directional bin count for all wind information on a neighborhood of the voxel. The final field is the Shannon entropy of every local histogram, computed across the whole data set [Xu et al., 2010]. Once the window shape and size is provided, we compute the histogram field in one go as we detailed in Sec. 5.5.2.

Results are reported in Fig. 5.8 where we show the hurricane (a rendering of its vapor density), its global directional histogram, and several entropy fields for box and Gaussian local neighborhoods using both the proposed method and a brute-

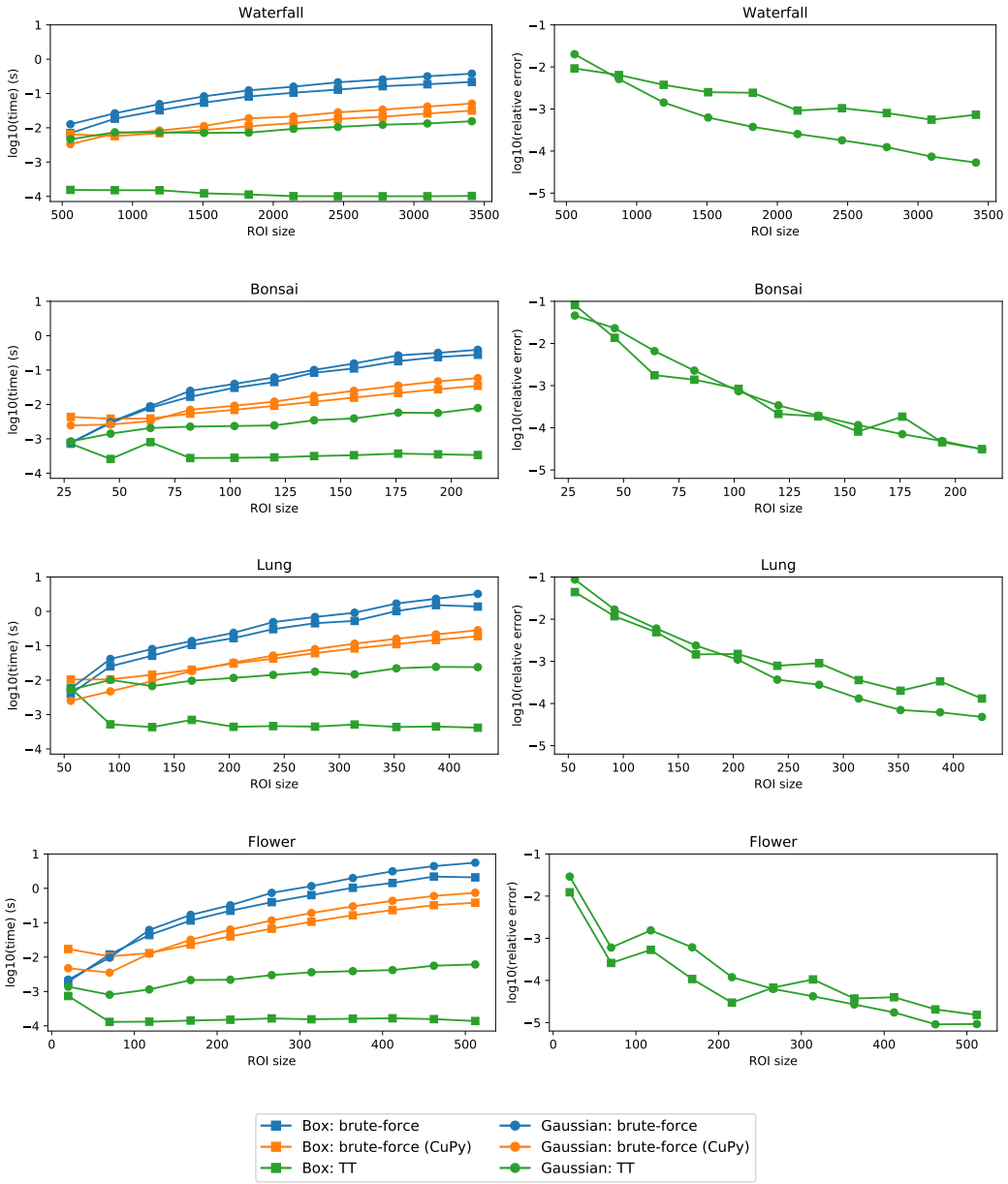


Figure 5.7: *Left: Average histogram reconstruction times for both box- and Gaussian-shaped regions. Right: Relative error of each reconstructed histogram. Our proposed TT representation generally outperforms brute-force approaches in terms of speed (even when they are CUDA-accelerated), and does so by an increasing margin as we increase the ROI size.*

force approach. The latter is especially prohibitive for non-rectangular regions; it works by computing one FFT multidimensional convolution per histogram bin. Our method achieves a manifold speed-up factor, also for the rectangular case, while its results deviate very little from the exact groundtruth values given by the brute-force.

5.6.4 Cross-Correlation Queries

A simple, yet effective approach for interactive visualization of high-dimensional vector fields, for example of histogram features, is user-driven transfer function selection. Essentially, a 1D array $\mathbf{w}$ of B weights is defined and its dot product with each histogram is computed; the resulting scalar is displayed through a suitable colormap. If $\mathbf{w}$ and the histograms are all normalized, the result is the normalized cross-correlation (NCC) between a template and all possible window neighborhoods.

As a last experiment we consider again the hurricane wind IH (Sec. 5.6.3) and obtain its NCC w.r.t. arbitrary windows within our proposed TT representation. We first precompute offline a histogram field over all neighborhoods of a fixed size as in the previous section, and then store the norm of each histogram. During interactive exploration we allow the user to define a template window W whose histogram $\mathbf{w}$ we extract (Secs. 5.4 and 5.5) and normalize. We then weigh the last TT core with $\mathbf{w}$ along its second dimension, i.e. use $\mathcal{B} \times_2 \mathbf{w}$ instead of $\mathcal{B}$, and finally reconstruct (Eq. 5.4) and divide by the precomputed norms. The resulting multiarray summarizes each window's histogram into one single scalar, namely the correlation between the window's directional histogram and that of the template W. This way, the user can highlight and identify regions whose local wind behavior is similar. Note that the NCC lies between 0 and 1 as our feature vectors (histograms) are non-negative. Fig. 5.9 shows visualization results for a number of different windows W of size $8 \times 8 \times 91$, with response times under 2 seconds.

5.7 Discussion

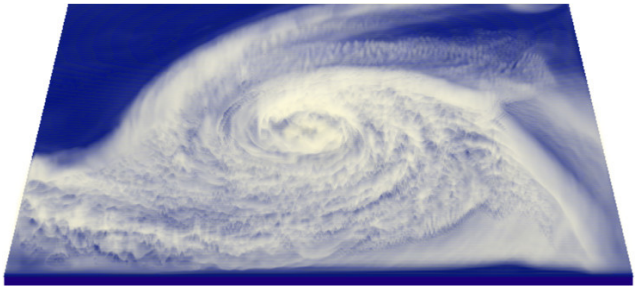
Based on these numerical experiments we observe that our proposed compressed representation takes up much less space than the conventional IH approach as shown in Table 5.2. This is useful when the uncompressed IH strains or exceeds the available computational and memory resources. The compression is lossy and comes at the expense of a variable compression error, see also Fig. 5.7. We observe that, in general, bigger data sets can be compressed better than smaller ones. The same applies for sparse data sets: all tensor slices that are filled with zeros (e.g. Bonsai or Flower) are represented in a TT compressed format with

zero-filled core slices, and in particular without increasing the TT ranks.

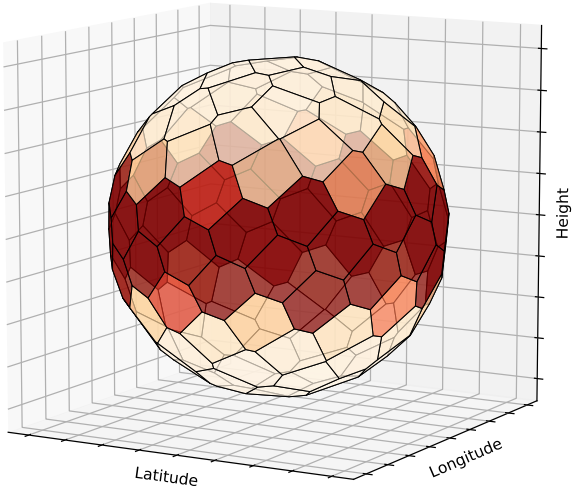
In terms of speed, TT reconstruction always becomes faster than naive brute-force traversal with ROIs of a certain size or larger. This overtaking point depends on the data set and prescribed ϵ, but usually is reached already by regions that are 100x or 1000x times smaller than the data set. As regards compression, our proposed incremental approach is robust and effective even for very large (512GB) integral histograms. Preprocessing times can take up to several hours, but such times are not uncommon for compression algorithms for large multidimensional data. If this time is taken into account, the break-even number of queries needed for our method to be overall faster than the naive traversal amounts to thousands (hundreds for histogram field reconstruction). However, we work under the asymmetry assumption, i.e. where interactive reconstruction is of paramount importance, especially when user interaction is involved.

Compared to wavelet-based alternative compression methods such as Wavelet-SAT [Lee and Shen, 2013], our method a) results in higher reduction rates (thanks to lossy compression and its use of adaptive transform bases); b) can reconstruct histograms over non-rectangular regions; and c) can apply transfer function weighing (and in particular, compute histogram field cross-correlations) at a small cost in the compressed domain. Its main drawback is of course the compression error introduced, but we found this relative error to be insignificant over medium to large ROIs, whereas the case of small ROIs is less important since brute-force traversal is actually the fastest method for such regions anyway.

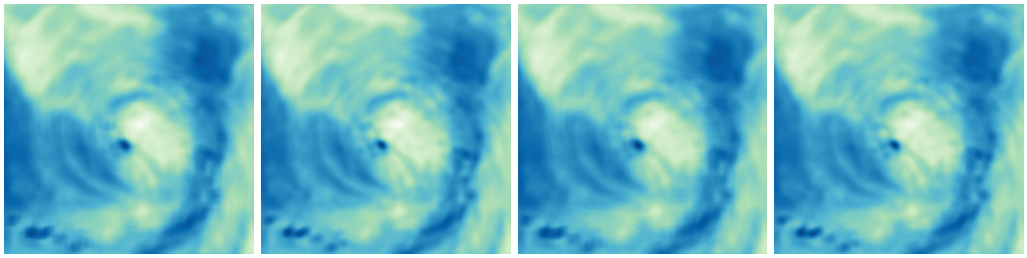
Our proposed reconstruction over Gaussian regions is significantly slower than its rectangular counterpart. However, it still outperforms the brute-force, usually even when exploiting GPU parallelism. We would like to highlight that, although general IHs are not well suited for non-rectangular regions, the tensor decomposition framework makes such queries possible thanks to its multilinearity and the range of compression-domain processing operations that it offers. The prescribed accuracy during compression is the main parameter of our system, and it results in a trade-off between smaller size and faster query time on one side versus lower error on the other side.



(a) Vapor density (volume rendering)



(b) Wind directional histogram, $B = 128$



(c) Box regions: brute-force (59.8s)

(d) Gaussian regions: brute-force (1316.4s)

(e) Box regions: TT (1.9s)

(f) Gaussian regions: TT (23.8s)

Figure 5.8: *(a) A* $500 \times 500 \times 91$ *hurricane vector field at its 25th timestep; (b) directional histogram for its (latitude, longitude, height) wind coordinates, grouped into 128 bins on the spherical surface; (c-f) entropy fields for both box- and Gaussian-shaped neighborhoods of size* $20 \times 20 \times 91$*, computed via brute-force vs. our proposed method (Sec. 5.5.2).*

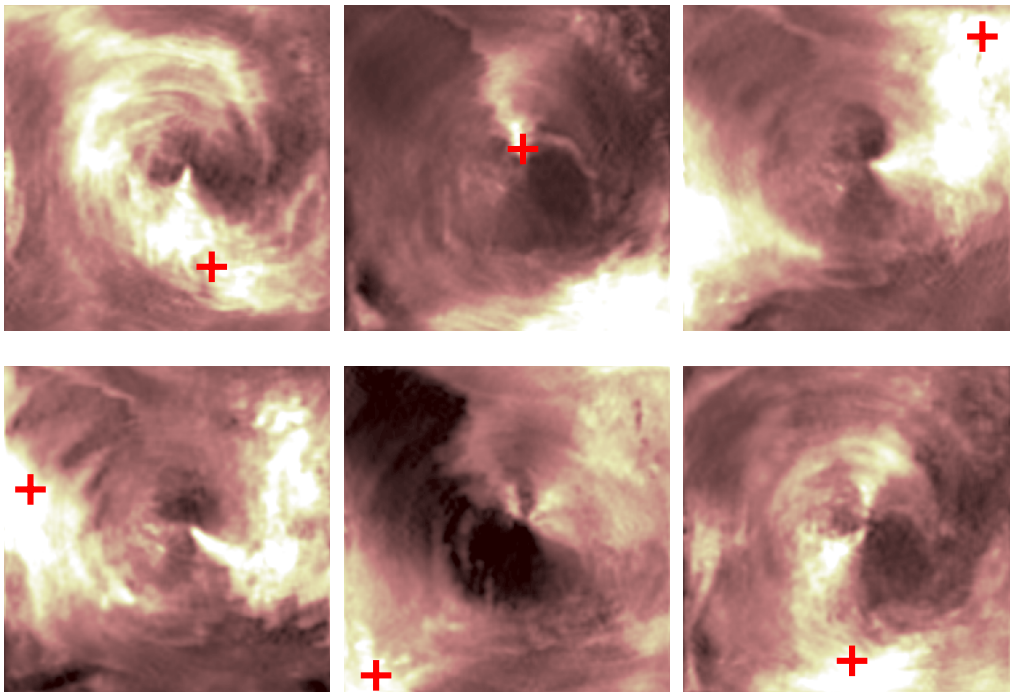


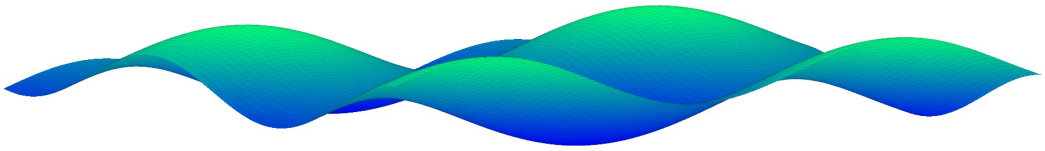
Figure 5.9: *Normalized cross-correlations between the hurricane wind vector field and six different template windows of size* $8\times8\times91$*. Brighter colors indicate higher histogram neighborhood correlations with a template whose center in each case is indicated by the red marker. This region-picking technique works in fact as a dynamic histogram transfer function selector that can e.g. highlight specific rain bands. Computation times ranged between 1.86s and 1.99s for these examples.*

Part II

Sparse Data: Interpolation and Learning

C H A P T E R 6

SURROGATE MODELING



6.1 Overview

Up to this point we have been concerned with dense tensors: full, entirely available multiarrays arising from high-resolution simulations and scanning devices. But often, one must deal with situations in which entire regions (or even the vast majority) of a tensor's domain of interest are unknown. The main goal is then to interpolate (learn) these missing regions, and often to do so implicitly, i.e. never handling the whole newly completed data in an explicit form. *Surrogate models* (also known as *meta-models* or *response surface models*) aim precisely for this. They approximate the true unknown model and can be cheaply evaluated. Sometimes they also allow for direct manipulation and computation of quantities of interest (QOI). Take for example a smooth 1D splines interpolation of a few (x, y) pairs. The interpolation's global maximum, among other properties, can be estimated efficiently from the model without having to sample it extensively.

Parameter spaces are prominent examples of incomplete tensors: they encode the set of all possible parameter combinations that can influence a simulation or experiment, along with their output (often, in the form of a single scalar value corresponding to a score). Interactive exploration and knowledge discovery of such spaces can greatly assist in the understanding of a simulation's behavior; however, they grow exponentially in size with the number of input parameters N (curse of dimensionality). Instead of precomputing and saving the interpolated data, surrogate models provide a routine to approximate any given element on demand. Therefore, once a good surrogate is available, the challenge is effectively shifted to the visualization domain. In this context, we aim for a tool that can be used for acquiring and compactly representing high-dimensional black-box spaces as well as for efficient visualization in the form of subspace selection, user-defined queries, on-demand statistics, etc.

Contribution

We tackle this sampling and visualization problem under the paradigm of tensor decompositions, and propose the TT format as a framework that provides a compact representation of the complete parameter space to operate with. Complex inter-variable dependences are encoded within the tensor ranks. The basic features of our framework include parameter tensor sampling and interactive reconstruction of arbitrary subspaces. The user can navigate the parameter space and visualize predictions in real time by moving a focus point. These capabilities are efficiently supported by the TT format, even when the dense parameter space contains billions of points. As an evaluation we showcase our framework with some examples of selection queries in parameter spaces using a global-to-local navigation strategy to browse different focus + context visualization diagrams.

The TT model and its unique properties are central to all stages of our pipeline, which is the first to display TT parameter spaces in an interactive analysis and visualization application.

Notation

Throughout this and the next chapters, a space of N parameters forms a Cartesian grid arising from a tensor product and has size $I_1 \times \cdots \times I_N$. We consider visualization diagrams that can be expressed as a set of M freely-moving indices, usually with $M \leq 3$. For example, a univariate plot is a subspace with $M = 1$, a surface plot (or an overlay of several 1D plots) has $M = 2$, etc.

We denote tuples of indices as $\boldsymbol{\alpha} \in \{0, 1\}^N$ and $\boldsymbol{\alpha} \subseteq \{1, \ldots, N\}$ interchangeably: a 0 (resp. 1) in the former notation means an index is absent (resp. present) in the latter. If a function $f : \mathbb{R}^4 \to \mathbb{R}$ only depends *effectively* on the two last variables, we may alternatively write $\boldsymbol{\alpha} = [0, 0, 1, 1]$ or $\boldsymbol{\alpha} = \{3, 4\}$, and $f_{\boldsymbol{\alpha}}(\mathbf{x}) \equiv f_{\boldsymbol{\alpha}}(\mathbf{x}_{\boldsymbol{\alpha}}) \equiv f_{3,4}(x_3, x_4)$ similarly to [Sudret, 2008], [Owen, 2014]. Cardinality of a set of variables is denoted as $|\boldsymbol{\alpha}|$, and coincides with the *Hamming weight* (bit sum) of its binary representation. Last, we write tuple complements as $-\boldsymbol{\alpha} \equiv \{1, \ldots, N\} \setminus \boldsymbol{\alpha}$.

6.2 Background

Visualization has long played a fundamental role in systematic analysis of dense data sets and parameter spaces [Chan, 2006], [Nocke et al., 2007], [Brecheisen et al., 2009], [Amirkhanov et al., 2010]. Interactive frameworks such as EnsembleVis [Potter et al., 2009], for example, deliver a collection of linked overview and statistical displays. The iLAMP method [dos Santos Amorim et al., 2012] uses multidimensional projection combined with interactive selection of points and subspaces in high-dimensional data sets. A related strategy by [Anand et al., 2012] exploits iteratively refined random projections to maximize the visual disparity in different subspaces. A recent interactive system for the segmentation of medical images [Pretorius et al., 2015] provides previews and thumbnails to assist in the estimation of good hyperparameters for segmentation algorithms. ParaGlide [Bergner et al., 2013] is a region-based parameter sensitivity analysis tool that partitions the parameter space into regions that represent distinct output behavior. This subdivision helps users in their understanding of qualitative differences among high-dimensional model outputs. This is similar in spirit to Morse-Smale complex-based approximations, which for example have been used together with dimensionality reduction techniques to produce a geometric summary of the space [Gerber et al., 2010]. [Tatu et al., 2012] use an interestingness-guided subspace search algorithm and various similarity functions to guide the

users during the visualization. For a compilation of further techniques for high-dimensional scientific visualization, see the survey [Kehrer and Hauser, 2013].

Many such visualization systems are highly domain-specific, and others rely on explicit (uncompressed) data representations. While some do evaluate surrogates on the fly during exploration, no prior analysis and visualization system exists that exploits the powerful tools made possible by tensor models.

6.3 Construction of TT Surrogates

The first key part of our pipeline is obtaining a high-quality TT interpolant. Fortunately, many models can be accurately represented by a low-rank TT model. For example, multiplicative functions (i.e. with the form $f_1(x_1) \cdots f_N(x_N)$) have exactly rank 1, while additive terms (i.e. $f_1(x_1) + \cdots + f_N(x_N)$) have exactly rank 2. More generally, we can build TT surrogates in a wide range of settings.

6.3.1 Preliminaries: Variable Range Discretization

In order to build the tensor product grid $I_1 \times \cdots \times I_N$ for our N-variable space, our TT surrogate $\tilde{f}(\mathbf{x}) \approx f(\mathbf{x})$ discretizes each variable's domain as a finite set $x_n(1) < \cdots < x_n(I_n)$. To record or evaluate an entry $\mathbf{x}$, each coordinate x_n must be first quantized to match the corresponding axis discretization. This is not a problem in practice, and discretizing the variable space is indeed a usual feature of several sampling strategies such as factorial design, Morris' method, one-at-a-time design, etc. [Iooss and Lemaître, 2015]. If needed, the grid can be refined by simply increasing the sampling resolution before building the surrogate, and all important TT operations have linear cost w.r.t. the spatial dimensions I_n. For simplicity we use nearest-neighbor interpolation to convert (quantize) an arbitrary real $\mathbf{x}$ to integer tensor indices $0, \ldots, I_n - 1$.

6.3.2 Construction From a Black-Box System

Black-box sampling is the scenario in which new samples $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_P\}$ are to be chosen and evaluated from scratch with no prior information on the inner workings of the true model. In other words, one has the freedom to define a set of samples $\mathbf{X}$ and run the simulation on it, and can do so adaptively, round by round, in order to minimize the model's generalization error. Adaptive cross approximation (ACA) builds a progressive sampling plan on the low-rank assumption; it is an example of design of experiments (DOE). ACA constructs the plan by progressively sampling certain *tensor fibers*: sets of samples obtained by fixing all parameters but one. This is a case of one-at-a-time sampling, which can improve

the DOE's overall efficiency (see also [Saltelli et al., 2008], 2.4.2). This guarantees that all possible discretized values for every variable are used at least once. Here we use an *alternating minimal energy* method to select the fibers [Dolgov and Savostyanov, 2014], an algorithm whose implementation has publicly been released as part of the Python ttpy toolbox [ttp,]. See Fig. 6.1 for an example of ACA structured sampling scheme in a 3D tensor.

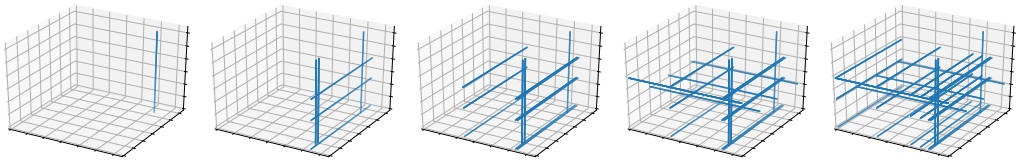


Figure 6.1: *Progression of the sampling plan during an ACA for a 3D tensor.*

6.3.3 From Categorical Data

Tensors are discrete data structures indexed by discrete axes and thus support categorical variables in a natural way (consider for example the 2D case: the rank of a matrix is not affected if we permute its columns and/or rows). Populating the missing entries of a tensor without any prior assumption about smoothness is known as *tensor completion* and is a very convenient tool for regression on categorical variables. It is similar to the better known problem of low-rank matrix completion for $N = 2$, but specific algorithms for $N \geq 3$ of course depend heavily on the particular decomposition format chosen (CP, Tucker, TT, etc.). We have implemented an *alternating least squares* (ALS) completion algorithm in the TT format, modified from [Grasedyck et al., 2013]. We use it to learn a 14D categorical data set (see Sec. 6.6.3). It is essentially a block coordinate descent, whereby one core is optimized at a time and the error is provably non-increasing. See App. A.2 for full details on the algorithm.

6.3.4 From an Auxiliary Regressor

More generally, one may want to interpolate the given training set first with a preferred regression method: support vector machines, radial basis functions, Gaussian processes, etc. One can then sample this auxiliary surrogate using ACA to build an approximate TT representation. This is a rather flexible approach and is feasible as long as the intermediate regressor can be approximated well by a surrogate of low TT ranks. Under this assumption, ACA works as a universal tool to reduce any model into the TT format, usable whenever an ad-hoc conversion in the compressed domain (such as the ones discussed next) is not available.

6.3.5 From Another Low-Rank Decomposition

Several well-known surrogate models are actually based on a low-rank expression, or can easily be cast as a low-rank format. We can convert from these cases more directly instead of relying on the general ACA as just discussed in Sec. 6.3.4.

From CP

TT ranks are bounded from above by CP ranks [Oseledets, 2011], and the proof is constructive: given a rank-R CP decomposition $[[\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]]$, an equivalent TT expression $[[\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N)}]]$ can be straightforwardly built as:

$$\mathcal{T}^{(n)}[x_n] := \begin{cases} \mathbf{U}^{(n)}[x_n, :] & \text{if } n = 1 \text{ or } n = N \\ \text{diag}(\mathbf{U}^{(n)}[x_n, :]), n = 1, \ldots, I_n & \text{if } 1 < n < N \end{cases} \tag{6.1}$$

where $\mathcal{T}^{(n)}[x_n]$ is a shortcut for $\mathcal{T}^{(n)}[:, x_n, :]$. Using this formula we can convert an arbitrary low-rank CP surrogate into a standard TT representation.

From Tucker

A TT can be also obtained from a Tucker decomposition, although TT ranks are not bounded by Tucker ranks (and vice versa). To do this conversion we start with the Tucker approximation formula

$$\widetilde{\mathcal{T}} = [[\mathcal{B}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]] \tag{6.2}$$

and then compress its core in the TT format:

$$\widetilde{\mathcal{T}} \approx [[\, [[\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(N)}]]; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]] \tag{6.3}$$

This can be also regarded a sequence of TT cores, each compressed by a matrix $\mathbf{U}^{(n)}$. By multilinearity, the right-hand side of Eq. 6.3 equals the following expression

$$[[\mathcal{B}^{(1)} \times_2 \mathbf{U}^{(1)}, \ldots, \mathcal{B}^{(N)} \times_2 \mathbf{U}^{(N)}]] \tag{6.4}$$

Eq. 6.3 is a so-called *TT-Tucker* decomposition, originally considered in [Oseledets and Tyrtyshnikov, 2010a]. This format generalizes Tucker, since it uses a (compressed) core and factor matrices, and also TT, as it is a sequence of (compressed) TT cores. In this sense, and given the major roles played by Tucker and TT in our work, TT-Tucker positively answers our second research question posed in Sec. 1.3. See Fig. 6.2 for a graphical representation in terms of tensor

networks, similarly to [Cichocki et al., 2016] (recall Sec. 2.5). The final TT cores are retrieved by explicitly performing the tensor-times-matrix operations:

$$\mathcal{T}^{(n)} = \mathcal{B}^{(n)} \times_2 \mathbf{U}^{(n)} \tag{6.5}$$

which increases the overall size, but is still linear w.r.t. N.

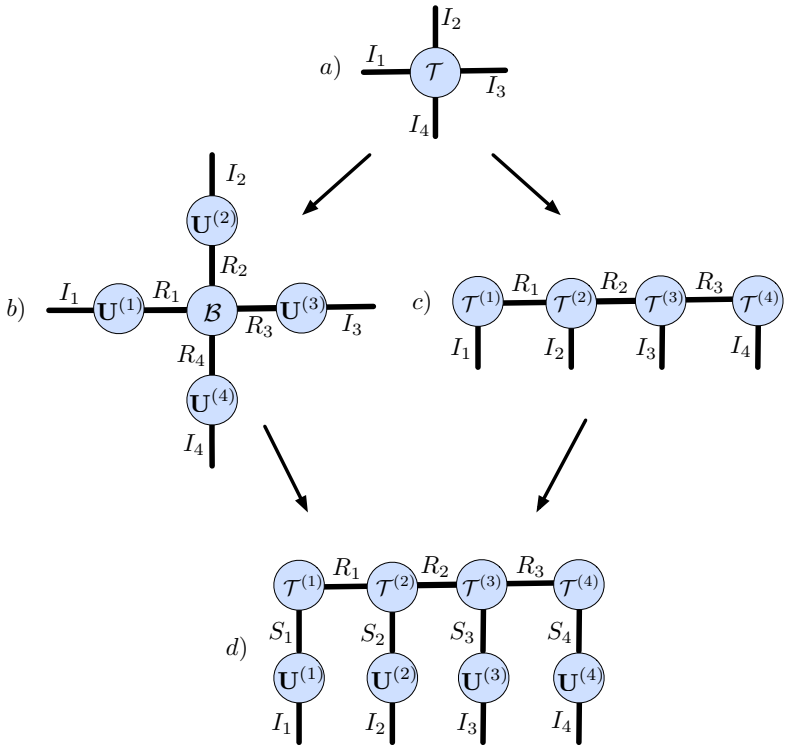


Figure 6.2: *a) A full 4D tensor $\mathcal{T}$; b) in Tucker format; c) in TT format; d) in the hybrid TT-Tucker format. The latter representation generalizes both Tucker and TT, and can be computed in two alternative but equivalent ways: either via TT compression of the Tucker core $\mathcal{B}$ (left path), or via Tucker compression (along the 2nd mode) of each individual TT core $\mathcal{T}^{(n)}$ (right path). Similarly, the Tucker format may be cast to TT by following either the path b)-a)-c) (full decompression and compression) or the much less expensive b)-d)-c).*

From Polynomial Chaos Expansions

PCE surrogate models [Soize and Ghanem, 2004] have been used in stochastic modeling and uncertainty quantification for decades. A PCE is based on a set of N polynomial bases $\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(N)}$ with each infinite basis $\mathcal{P}^{(n)} = \{\mathcal{P}_0^{(n)}, \mathcal{P}_1^{(n)}, \dots\}$ being orthogonal w.r.t. a marginal PDF dF_n over a real interval Ω_n:

$$\int_{\Omega_n} \mathcal{P}_i^{(n)}(x_n)\mathcal{P}_j^{(n)}(x_n)\, dF_n(x_n) = 0 \;\forall n \text{ iff } i \neq j \tag{6.6}$$

The PCE of bounded degree D approximates a function $f : \Omega = \Omega_1 \times \cdots \times \Omega_N \subset \mathbb{R}^N \to \mathbb{R}$ as a truncated expansion in terms of these bases:

$$f(\mathbf{x}) \approx \sum_{\boldsymbol{\alpha}=(0,\ldots,0)}^{(D,\ldots,D)} f_{\boldsymbol{\alpha}} \cdot \Psi_{\boldsymbol{\alpha}}(\mathbf{x}) \tag{6.7}$$

with

$$\Psi_{\boldsymbol{\alpha}}(\mathbf{x}) := \prod_{n=1}^{N} \mathcal{P}_{\alpha_n}^{(n)}(x_n) \tag{6.8}$$

We can convert any such PCE representation into a TT surrogate as follows, at the expense of only a small discretization error that can be easily adjusted (recall Sec. 6.3.1). Eq. 6.7 is interpretable as a *continuous* Tucker decomposition, with the $f_{\boldsymbol{\alpha}}$ acting as the elements of a core of size $(D+1)^N$. To obtain a standard Tucker format we just need to define its factor matrices. Each factor $\mathbf{U}^{(n)}$ has size $I_n \times (D+1)$ and is found by sampling the corresponding polynomial basis over the discretized variable range $x_n(1), \ldots, x_n(I_n)$:

$$\mathbf{U}^{(n)}[i,j] := \mathcal{P}_j^{(n)}(x_n(i)) \tag{6.9}$$

After this we can apply the conversion method detailed in Sec. 6.3.5 to get the equivalent TT representation.

Alternatively, we may also convert a low-rank PCE expansion [Konakli and Sudret, 2016] to TT by means of the CP conversion method above. Such low-rank expansions define a *continuous* CP as

$$f(\mathbf{x}) \approx \sum_{r=1}^{R} \lambda_r \cdot \Psi_r(\mathbf{x}) \tag{6.10}$$

with each rank-1 component arising from N functions that admit a low-degree polynomial expansion:

$$\Psi_r(\mathbf{x}) := \prod_{n=1}^{N} \left(\sum_{d=0}^{D} \alpha_{rd}^{(n)} \cdot \mathcal{P}_d^{(n)}(x_n) \right) \tag{6.11}$$

and can be converted into standard PCE via discretization, analogously to Eq. 6.9.

6.4 Visualization in the TT Format

6.4.1 Reconstructing Compressed Subspaces

Standard visualization diagrams such as 1D and surface plots, tables, and combinations thereof (hierarchical axis, dimensional stacking, overlaid plots, etc.) can be viewed as discretized axis-aligned subspaces of $\mathcal{T}$. It is straightforward to fix or restrict the movement of multilinear indices within any tensor decomposition. In the TT format in particular, we can set the n-th parameter to its k-th value by substituting the tensor core $\mathcal{T}^{(n)}$ with a $R_{n-1} \times 1 \times R_n$ core, namely $\mathcal{T}^{(n)}[:, k, :]$. The asymptotic reconstruction cost is $O(I^k R + R^2)$, where k is the number of free parameters. We decompress such tensors by a sequence of matrix products and multiarray reshaping operations, similarly to the Tucker model. Since our simulations' behavior can be usually captured accurately using only a few TT ranks, in our experiments the subspace reconstruction cost was always in the order of a few milliseconds.

6.4.2 From Tensor Train to Parallel Coordinates

In order to further illustrate the versatility of such a TT parameter representation, we detail here how to obtain a parallel coordinates diagram from our compressed tensor $\mathcal{T}$. While this type of diagrams is applied to sparse data mostly, one can extend it to a dense space by varying the polyline styling. We make the opacity of each polyline proportional to the value at its corresponding position in the tensor (in the spirit of density-based parallel coordinates [Heinrich and Weiskopf, 2013]). We assume linearity, so the total opacity of a segment equals the sum of the opacities of all polylines that include it. Thus the opacity of all $I_n I_{n+1}$ segments between two adjacent coordinates n and $n+1$ in the diagram is determined by the sum of $\mathcal{T}$ along all indices fixing the n-th and $(n+1)$-th ones. This sum yields an $I_n \times I_{n+1}$ matrix $\mathbf{S}$: for each entry (i, j), a segment with opacity $\mathbf{S}[i, j]$ connecting abscissas i and j is to be drawn in the diagram. We compute this matrix by projecting (summing) $\mathcal{T}$ along all dimensions but n and $n+1$. Note that in the TT format this only requires summing the affected cores along their second mode. In total we have to display $(N-1)I^2$ segments and perform $O(NI^2R^2)$ operations, as opposed to the $O(NI^N)$ operations that a naive traversal of the full uncompressed tensor would take. The steps are summarized in Alg. 7.

Note that the pairs $(n, n+1)$ traverse the dimensions in their original order, but any reordering of coordinates is possible if we sum over the appropriate sets of indices.

Algorithm 7 Plot a parallel coordinates diagram from a surrogate TT $\mathcal{T}$ with ranks $R_1, \ldots, R_{N-1}$.

for n in $1, \ldots, N-1$ **do**
  for m in $1, \ldots, N$ **do**
    $\mathcal{C}^{(m)} := \sum_{i_m=0}^{I_m-1} \mathcal{T}^{(m)}[:, i_m, :]$ {Projection along the m-th mode, cast as a tensor of shape $R_{m-1} \times 1 \times R_m$}
  end for
  $\mathcal{C}^{(n)} := \mathcal{T}^{(n)}$
  $\mathcal{C}^{(n+1)} := \mathcal{T}^{(n+1)}$
  $\mathcal{C} := [[\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(N)}]]$ {$\mathcal{C}$ has size $1 \times \cdots \times I_n \times I_{n+1} \times \cdots \times 1$}
  $\mathbf{S} := \mathrm{decompress}(\mathcal{C})$ {$\mathbf{S}$ has size $I_n \times I_{n+1}$}
  for i in $1, \ldots, I_n$ **do**
    for j in $0, \ldots, I_{n+1} - 1$ **do**
      $\mathrm{source} := (n, i)$
      $\mathrm{target} := (n+1, j)$
      $\mathrm{opacity} := \mathbf{S}[i, j]$
      $\mathrm{drawSegment}(\mathrm{source}, \mathrm{target}, \mathrm{opacity})$
    end for
  end for
end for

6.4.3 Bivariate Projections

In Alg. 7 the projections onto all consecutive pairs of dimensions $(n, n+1)$ are computed. It is straightforward to compute the projection of $\mathcal{T}$ onto every other possible pair of parameters. Each projection forms a 2D array (image), and one can arrange the $N^2 - N$ possible images as a matrix with an empty diagonal. Note that, if one normalizes the colormap to the data range, displaying projections is equivalent to displaying averages.

6.4.4 Finding Interesting Subspaces

Users often ask themselves in what parameter regions the surrogate's output is especially responsive. We tackle here the problem of finding the axis-aligned k-dimensional subspace that has the highest (or smallest) possible variance. Note that the solution space is large as there are $\binom{N}{k} \cdot I^{N-k}$ possible such subspaces within an ambient tensor of size I^N. We cast this as a global optimization problem in the TT format, which we then solve using ACA as outlined back in Sec. 2.7.3. First of all let us introduce the *Hamming mask tensor* of order k, denoted as $\mathcal{M}^k$, which has size 2^N. For each entry $\boldsymbol{\alpha} \in \{0, 1\}^N$,

$$\mathcal{M}^k[\boldsymbol{\alpha}] := \begin{cases} 1 & \text{if } \boldsymbol{\alpha} \text{ has } k \text{ non-zeros} \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

This mask will play a role in the next chapter as well. We are able to build $\mathcal{M}^k$ in a compressed form using only $k+1$ ranks (Fig. 6.3). It is best understood as a deterministic finite automaton that reads N symbols out of the alphabet $\{0, 1\}$. Each core slice is the state transition matrix corresponding to one of the two possible input symbols. Reconstructing one element from a TT is equivalent to taking a vector and multiplying it by a sequence of matrices. The vector at the n-th step has size $k+1$ and encodes how many non-zeros have been encountered so far: a '1' at its first position means none, a '0' followed by a '1' means one, etc. The core slices transform this vector counter to account for the new symbols as we traverse the sequence. The first slice of each core is the identity matrix, since it corresponds to the symbol 0 (which does not alter the counter of non-zeros). The other slices, however, must increment the counter, i.e. shift the '1' one position towards the right. They are therefore implemented as a shifted identity matrix. The last core simply checks if the total number of non-zeros found until the end matches k or not.

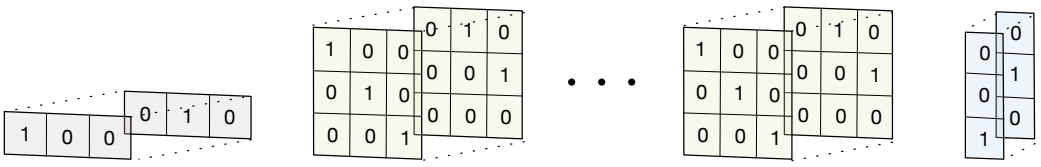


Figure 6.3: *The Hamming mask tensor train $\mathcal{M}^k$ for order $k = 2$. At each position $\boldsymbol{\alpha} \in \{0, 1\}^N$ it contains a '1' if and only if $|\boldsymbol{\alpha}| = k$, and 0 otherwise. It is compressed with N cores (rank $k+1$) using $2(k+1)^2(N-2) + 4(k+1)$ elements in total.*

We now proceed as follows: we build a TT $\mathcal{V}$ that compactly encodes the variances for *all* possible subspaces of any dimensionality $0 \leq k \leq N$. Every position $\mathcal{V}[\mathbf{x}]$ encodes one such subspace: each zero index $x_n = 0$ means that the corresponding dimension n is one of the k free dimensions of the subspace, while the remaining indices $x_n \geq 1$ determine the $N-k$ anchor point position. Let $\bar{\mathcal{M}}^{N-k}$ be a TT tensor of size $(I_1+1) \times \cdots \times (I_N+1)$ built as follows: for each core n, its first slice is the first slice of the n-th core of $\mathcal{M}^{N-k}$, whereas the remaining ones are the second slice repeated I_n times. We then multiply $\mathcal{V}$ element-wise with $\bar{\mathcal{M}}^{N-k}$; this filters out all subspaces that have dimension other than k. Finally we just find the global maximum of the resulting tensor. See the full details in Alg. 8; this restrict-and-search strategy will also prove useful in Ch. 7.1.

Algorithm 8 Discover the axis-aligned subspace of dimension k with maximal variance. To this end we write the variance in two parts: $\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}^2[X]$.

{First part $\mathcal{T}_1 = [[\mathcal{T}_1^{(1)}, \ldots, \mathcal{T}_1^{(N)}]]$: raw second moment of all possible subspaces}
$\mathcal{S} := \mathcal{T}^2 = \mathcal{T} \circ \mathcal{T}$ {Element-wise square}
for n in $1, \ldots, N$ **do**
  $\mathcal{T}_1^{(n)}[:, 0, :] = \frac{1}{I_n} \cdot \sum_{i_n=1}^{I_n} \mathcal{S}^{(n)}[:, i_n, :]$ {First slice: the mean of $\mathcal{S}$}
  $\mathcal{T}_1^{(n)}[:, 1{:}I_n + 1, :] = \mathcal{T}[:, 1{:}I_n + 1, :]$ {Remaining slices: copy of $\mathcal{S}$}
end for
{Now, similarly for the second part $\mathcal{T}_2 = [[\mathcal{T}_2^{(1)}, \ldots, \mathcal{T}_2^{(N)}]]$}
for n in $1, \ldots, N$ **do**
  $\mathcal{T}_2^{(n)}[:, 0, :] = \frac{1}{I_n} \cdot \sum_{i_n=1}^{I_n} \mathcal{T}^{(n)}[:, i_n, :]$
  $\mathcal{T}_2^{(n)}[:, 1{:}I_n + 1, :] = \mathcal{S}[:, 1{:}I_n + 1, :]$
end for
$\mathcal{V} := \mathcal{T}_1 - \mathcal{T}_2$ {Variances of all possible subspaces}
$\mathcal{V} := \mathcal{V} \circ \bar{\mathcal{M}}^{N-k}$ {Set to zero all subspaces that do not have dimensionality k}
return $\mathbf{s} := \arg\max\{\mathcal{V}\}$ {Index of N elements}
{$\mathbf{s}$ encodes the best subspace: its k free dimensions are marked as k zero entries. Its non-zero entries encode the $(N-k)$-dimensional anchor point that the subspace passes through}

6.5 User Interaction

The previous sections have covered the building blocks of our navigation system: sampling paradigms and efficient analysis and visualization tools that can be leveraged from a TT decomposition. Here we describe the interaction elements available for the user for the overall knowledge discovery process.

Data Acquisition

The sampling stage is straightforward for the user, as it always follows three sequential steps:

1. Specify a numerical simulation to interface with, or a fixed set of data samples.

2. Define the space: number of parameters and sampling density along each axis.

3. Run a suitable model fitting procedure (ACA or otherwise) with chosen accuracy.

Analysis and Interactive Exploration

There are two main diagram types: the ones that display a general summary of the space (global visualization) and the ones that allow a local-to-global visualization strategy.

- Global visualization:
  - Show a parallel coordinates diagram (Sec. 6.4.2). The ordering of dimensions can be varied, and each parameter's movement can be restricted to a subinterval.
  - Show uni- or bivariate projections (Sec. 6.4.3).
  - Analysis: show global extrema, interesting subspaces
- Localized navigation: a series of subspace tensors are shown, each grouping one or more parameters together. The navigation window is centered on one focus point, signaled by vertical bars for 1D diagrams and red markers for 2D diagrams). Each diagram shows how the simulation will behave when its parameter(s) is/are moved while the rest are fixed on the focus. The focus point can be interactively modified by displacing its bars and markers.

6.6 Results

We have tested the proposed visualization criteria on a number of settings. We use Python 3.5 and exploit the ttpy toolbox [ttp,], a Python/Fortran library for TT manipulation that supports, among others, compression from full explicit tensors, slicing, decompression, truncation (rounding), and cross-approximation for any dimensionality.

6.6.1 Synthetic Simulation

We defined the following 5-parametric synthetic model as a first example:

$$f(x,y,z,w,t) = \begin{cases} x + 10 \cdot \sin(y) - z^2/100 & \text{if } 20 \leq w \leq 30 \\ 0 & \text{otherwise} \end{cases}$$

Note that t is a non-essential dimension as it has no influence on the model output. Our grid discretization is $(1,\ldots,50)$ for $n = 1,\ldots,5$, so the resulting tensor product space has $\prod_{i=1}^{N} I_n = 50^5 \approx 3.1 \cdot 10^8$ elements, out of which the

ACA method sampled 26650 elements to produce a compressed TT-tensor with ranks $(2, 2, 2, 2)$ and 800 double-precision values. Due to this synthetic model being composed of elementary mathematical functions, the ACA approximation achieves a relative error $\epsilon < 10^{-15}$ over a test set of 4096 samples drawn via Latin hypercube sampling (LHS, [McKay et al., 1979]) on the domain.

The parallel coordinates diagram in Fig. 6.4 gives a good macroscopic overview of each variable's influence, e.g. the linear contribution from x, the sinusoidal pattern of y, and the bandpass condition imposed by w. Computing the opacities for the $\sum_{n=1}^{N-1} I_n I_{n+1} = 10000$ segments from the compressed tensor took only 6.0ms.

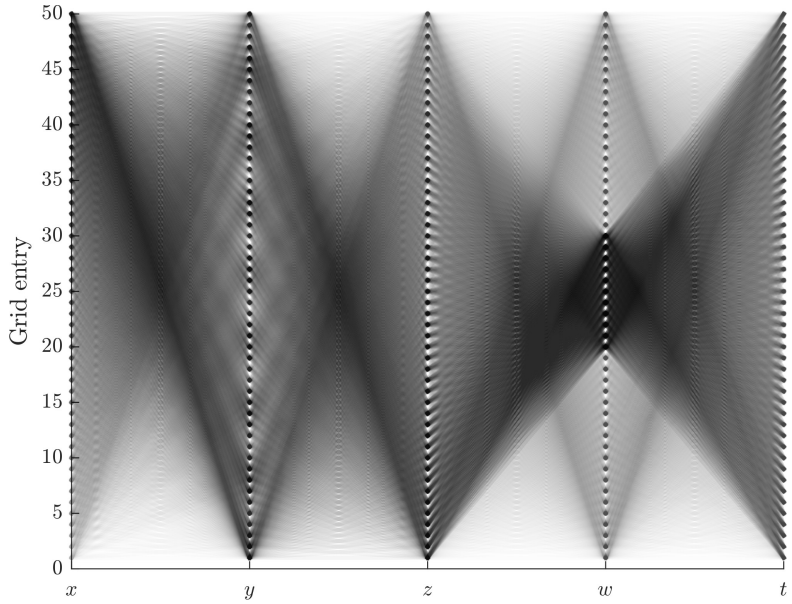


Figure 6.4: *Parallel coordinates diagram for the synthetic simulation, directly extracted from the compressed $\mathcal{T}$.*

6.6.2 Saint-Venant Flood Model

This model [Lamboni et al., 2013] estimates the maximum annual river height H, the water overflow S, and the cost C of a dike as a function of 8 parameters (see Tab. 6.1):

$$\begin{cases} H = \left(\frac{Q}{BK_s \sqrt{(Z_m - Z_v)/L}} \right)^{(3/5)} \\ S = Z_v + H - H_d - C_b \\ C = 1_{S>0} + 1_{S \leq 0} \cdot (0.2 + 0.8(1 - e^{-1000/S^4})) + 0.05 \cdot \min(H_d, 8) \end{cases} \tag{6.13}$$

Table 6.1: *Parameters of the Saint-Venant flood model*

Variable	Description	Units	Range
Q	Maximal annual flow	m^3/s	$[500, 3000]$
K_s	Strickler coefficient	$m^{1/3}/s$	$[15, 100]$
Z_v	River downstream level	m	$[49, 51]$
Z_m	River upstream level	m	$[54, 56]$
H_d	Dike height	m	$[7, 9]$
C_b	Bank level	m	$[55, 56]$
L	Length of river stretch	m	$[4990, 5010]$
B	River width	m	$[295, 305]$

We choose C (total cost of the project measured in millions of euros) as the output to predict, i.e. $f(\mathbf{x}) := C$. Note that it is a piece-wise defined and non-differentiable function. We discretize the grid using $I = 128$ bins per dimension and build our model with ACA using 132224 function evaluations (maximum rank $R = 9$), which achieves $\epsilon \approx 0.013\%$ on a test set of 4096 sample points (again, drawn using LHS).

Fig. 6.5 shows a *Trellis display*, another example of suitable tensor diagram. Such displays are similar to the related technique known as *dimensional stacking*: two dimensions are nested inside another two. The two outer ones move in a limited set of values, while the inner ones move in their full range. Trellis displays are able to deliver global contents of a tensor up to 4 dimensions. Since all slices are aligned and parallel to the axes, reconstruction from a TT is extremely fast. In Fig. 6.5 the tensor was projected (summed) along 4 dimensions.

Fig. 6.6 shows the global maximum of the space (which took 2.9 seconds to estimate) and the predictions along fibers and surfaces that pass through it. Two examples of interesting 2D subspace selection (Sec. 6.4.4) are given in Fig. 6.7. Since the ambient space has size 128^8, there are $\binom{8}{2} \cdot 128^{8-2} \approx 1.23 \cdot 10^{14}$ possible surfaces.

6.6.3 GEMM Matrix Product in the GPU

Our last experiment is a parallel computing example: we measured the computation time of 32-bit floating point matrix-matrix products in a graphics processing unit (GPU) according to 14 parameters and optimization techniques (loop unrolling, thread block-size, vector data types, etc.). The input variables are essentially discrete, since they are highly non-linear [Nugteren and Codreanu, 2015] and can take only a handful of different values at most (usually a few powers of 2). We have chosen to build our TT surrogate using tensor completion as we described in Sec. 6.3.3. The specific algorithm is dynamic ALS (see full details in

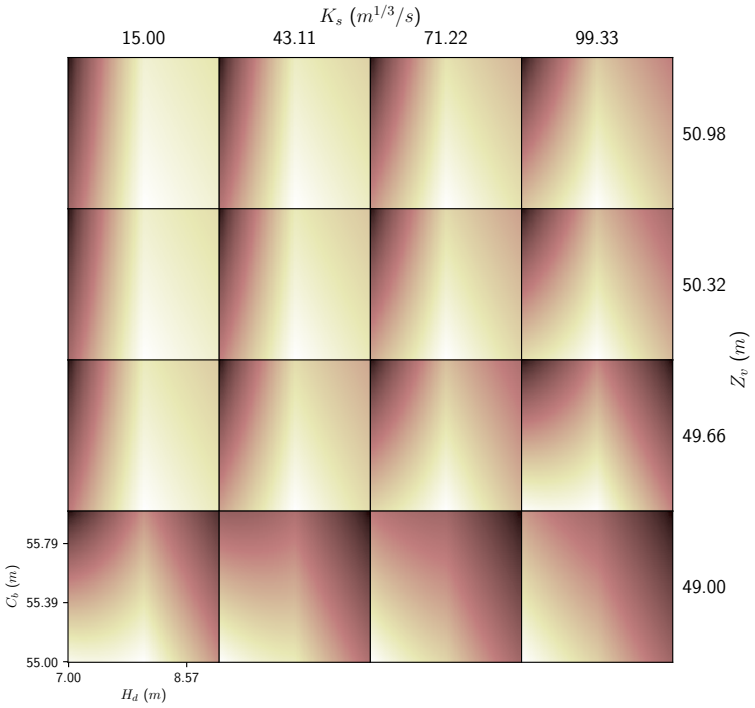


Figure 6.5: *Trellis display of our Saint-Venant surrogate model. The model is summed up along the four dimensions that are not shown. Such diagrams are excellent at delivering complex interactions across the whole domain of variables. The summation took* 5.8*ms, while reconstructing the* 16 *slices (*$2.6 \cdot 10^5$ *points in total) took* 11.4*ms in total.*

App. A.2). The product analyzed is $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ with all three matrices having size 2048×2048. We use the highly-tuneable GEMM kernel provided in the package CLTune [Nugteren and Codreanu, 2015], a generic auto-tuner for OpenCL kernels written in C++. Tab. 6.2 summarizes the 14 parameters and their input ranges (see the CLTune paper for further details).

We generated this data set with a workstation running Ubuntu Linux 16.04, equipped with an Intel Core i5-4690 3.5GHz processor and a GeForce GTX680 GPU with 4GB of memory. The data set consists of 12080 samples taken uniformly at random (without repetition) among the 1327104 total possible variable combinations. Each sample was measured 25 times and averaged in order to reduce noise effects. All GEMM running times are considered in logarithmic scale both for training and analysis as advised in [Falch and Elster, 2017]. We split the data as 70%, 15%, and 15% for training, validation and test, respectively. The best TT surrogate was obtained after 25 ALS iterations, which took 28.7 seconds. It has ranks $R_1 = \cdots = R_{13} = 8$ for a total of 2224 non-zero elements, and it

Table 6.2: *The 14 parameters of the GEMM OpenCL kernel*

Variable(s)	Description	Domain
M_{wg}, N_{wg}	Per-matrix 2D tiling at workgroup level	$\{16, 32, 64, 128\}$
K_{wg}	Inner dimension of 2D tiling at workgroup level	$\{16, 32\}$
M_{dimC}, N_{dimC}	Local workgroup size	$\{8, 16, 32\}$
M_{dimA}, N_{dimB}	Local memory shape (when enable)	$\{8, 16, 32\}$
K_{wi}	Kernel loop unrolling factor	$\{2, 8\}$
M_{vec}, N_{vec}	Per-matrix vector widths for loading and storing	$\{1, 2, 4, 8\}$
M_{stride}, N_{stride}	Enable stride for accessing off-chip memory within a single thread	$\{yes, no\}$
$L\$_A, L\$_B$	Per-matrix manual caching of the 2D workgroup tile	$\{yes, no\}$

achieved a relative error of $\epsilon \approx 3.4\%$ on the test set (see Fig. 6.8).

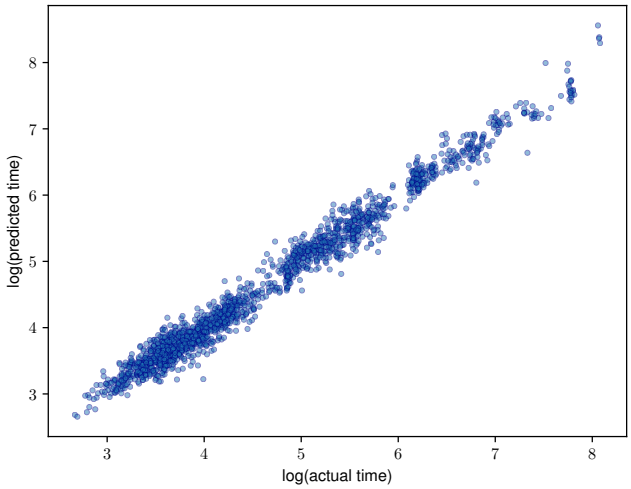
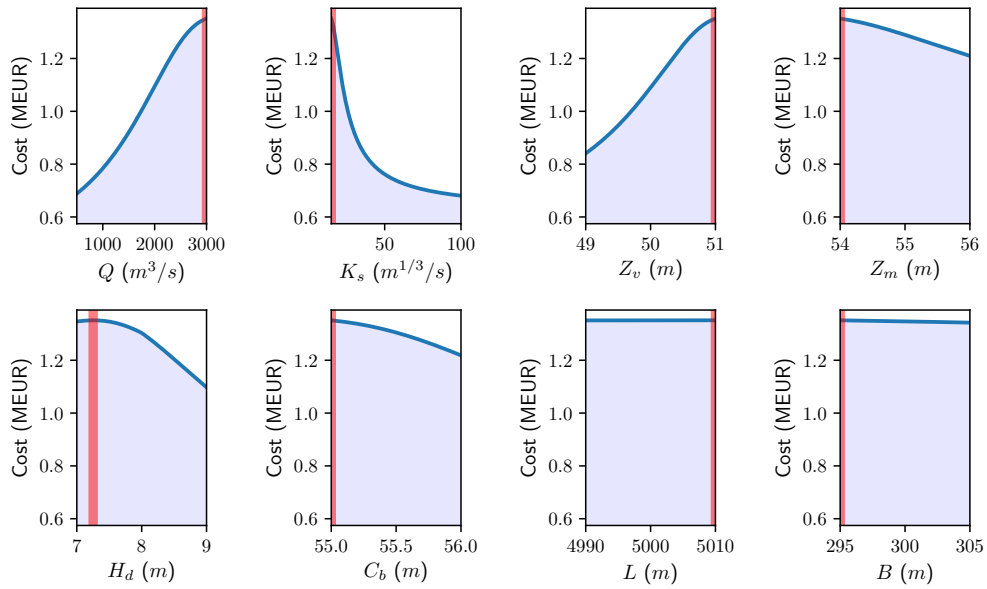


Figure 6.8: *Surrogate obtained via TT completion for our GEMM experiment: groundtruth vs. prediction over the test set (1812 points), with relative error* $\epsilon \approx 3.4\%$

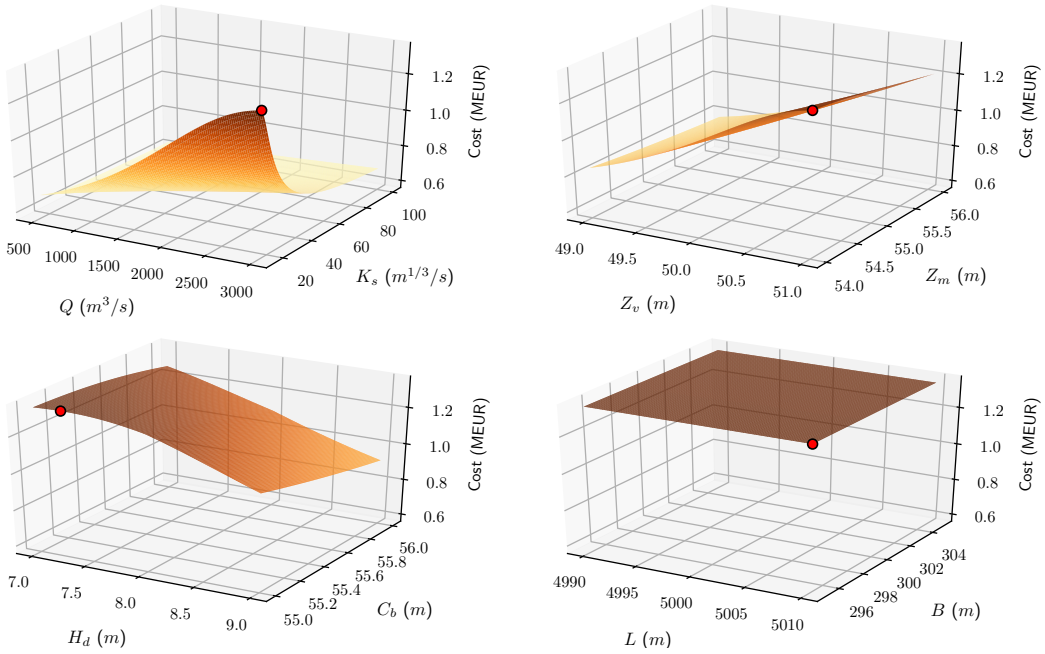
Fig. 6.9 shows a Trellis display for this categorical data set. Note that such displays are able to show the full domain in the case of categorical variables, since they do not require discretization.

6.7 Discussion

We have described a visualization system that creates and operates on a surrogate model in order to predict and help analyze the behavior of scientific simulations. Cross-approximated sampling algorithms, in particular, relieve the user from tedious trial-and-error explorations, while tensor decompositions offer many possibilities for manipulating and visualizing multidimensional data. Thanks to the TT model one can combine these two aspects into one single framework. In our system, the output of the simulation with respect to its parameters was considered as a high-dimensional space and compressed into a low-rank tensor. TT constructions scale linearly with the number of parameters and the amount of sampled points. We have shown how the user can select and effectively reconstruct interesting subspaces from the compressed domain in real time and interactively. Several types of visualization diagrams and techniques can benefit from the compactness and efficiency of the TT framework.

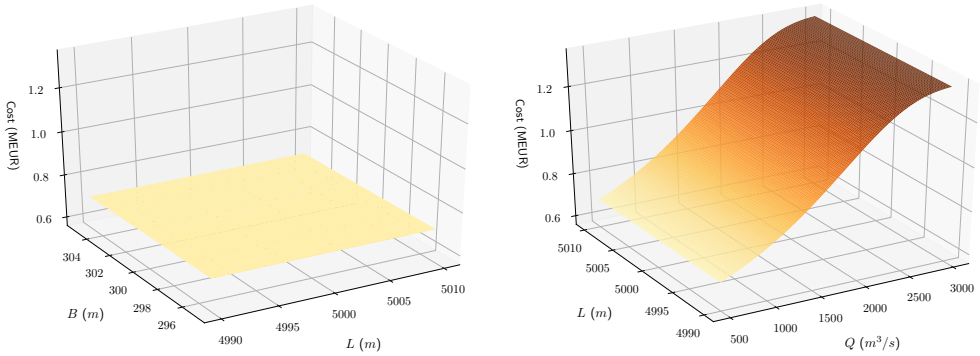


(a) Fiber-based



(b) Surface-based

Figure 6.6: *Navigation of the Saint-Venant model. The draggable focus point is shown as vertical bars for the fiber plots and as circle markers for the surface plots. It is currently set as the global maximum of the tensor and can be moved interactively.*



(a) Lowest variance $\approx 2.36 \cdot 10^{-6}$ (b) Highest variance ≈ 0.0487

Figure 6.7: *Axis-aligned surfaces with the lowest and highest variance in the Saint-Venant surrogate model. The first took* 4.81 *seconds and passes through* $Q = 2133.86, K_s = 15, Z_v = 50.40, Z_m = 54, H_d = 9, C_b = 56$*, while the second took* 3.98 *seconds and passes through* $K_s = 15, Z_v = 51, Z_m = 54, H_d = 7, C_b = 55, B = 295$*.*

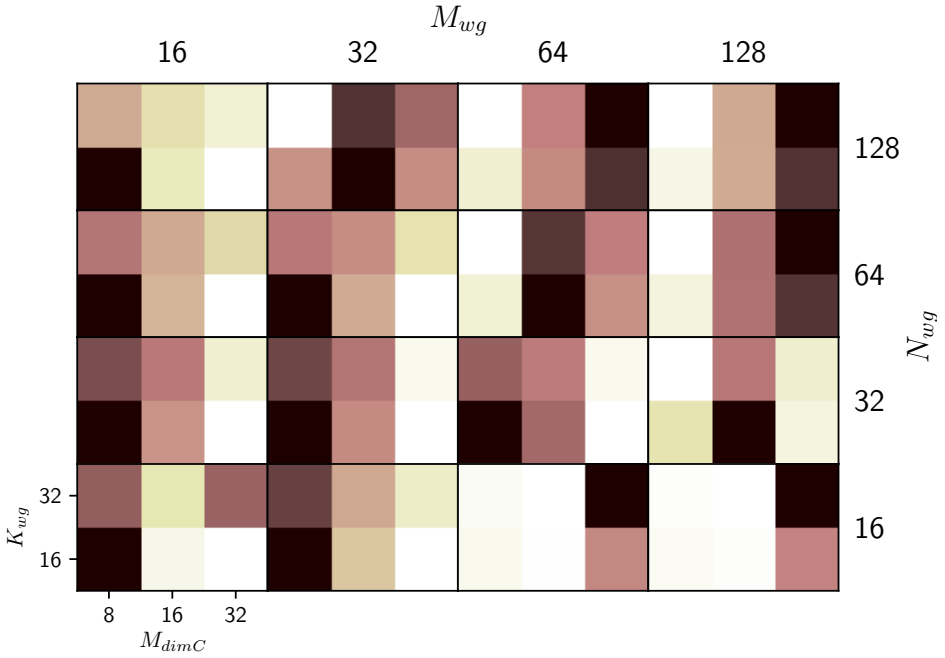
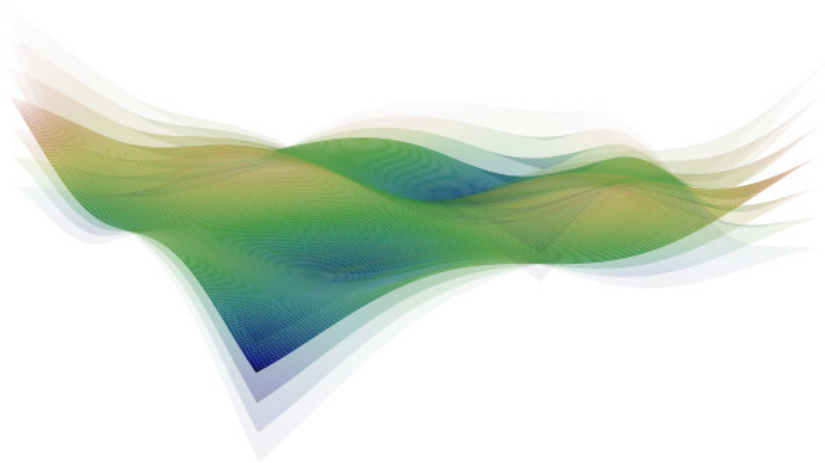


Figure 6.9: *Trellis display for our GEMM completed tensor (whiter is better, i.e. lower predicted GEMM running times). Again, dimensions not shown are averaged.*

CHAPTER 7

SENSITIVITY ANALYSIS



7.1 Overview

In the previous chapter we have proposed several ways to build and visualize TT surrogate models. In this last technical chapter we turn to the analytical side of tensor-learned computational models and physical simulations, in particular to assessing the influence of each input variable (and all combinations thereof) on the model's output. The quantitative study of such influences is known as *sensitivity analysis* (SA). When the variables are stochastic, the propagation of their uncertainty towards the model output must also be taken into account. We focus on variance-based SA, often referred to as *analysis of variances* (ANOVA), and in particular the so-called *Sobol decomposition*. It approximates the parametrized model as a sum of simpler functions, each depending on only a subset of the original set of variables. The sensitivity to each variable is then reflected by the functions that depend on it, and can therefore be estimated as their relative contribution to the output's overall statistical variance. These relative variances have become a standard tool for global SA in the last few decades [Saltelli et al., 2008], [Sudret, 2008], [Marrel et al., 2009], [Owen et al., 2013], [Iooss and Lemaître, 2015].

A popular method to compute such variance indices is via *Monte Carlo* (MC) integration estimators on a suitable set of samples within the variable space (the *sampling plan*). This was already outlined in Sobol's original paper [Sobol, 1990] and has gained momentum thereafter. However, MC convergence is slow w.r.t the number of samples available [Iooss and Lemaître, 2015]. Structured sampling plans exist that improve convergence, e.g. *Latin hypercube sampling* or *quasi-random sequences* (quasi-MC). If needed one may favor estimators for *total effect* indices, i.e. *quantities of interest* (QOI) that aggregate indices of diverse orders together. Unfortunately, a plan tailored to estimate a particular index, or set thereof, may be suboptimal for other indices. In practice, analysts often choose to restrict the Sobol decomposition to interactions of low-order (e.g. up to 2), and/or perform a prior dimensionality reduction in what is known as *screening* (e.g. freezing seemingly unimportant variables). Such simplifications greatly reduce the computational complexity, but pose a risk: they might fail to detect significant complex interactions between variables, and over-zealous reduction can harm subsequent processing steps in the analysis pipeline.

A complementary approach to direct MC estimation is building a surrogate model in an offline preliminary step. The surrogate acts as an interpolator that is fast to evaluate and can approximate the true unknown model at arbitrary sampling points [Queipo et al., 2005]. This strategy is attractive when sample acquisition is expensive or highly dynamic, especially if the analyst would like to estimate new indices on demand. Furthermore, several surrogates can produce Sobol indices in a more direct manner [Iooss and Lemaître, 2015], thus avoiding MC integration altogether. However, dealing with high-dimensional parametric systems, i.e. with

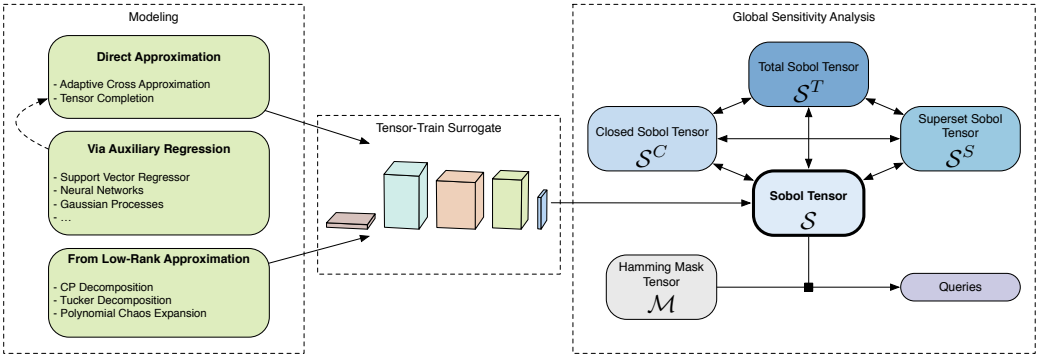


Figure 7.1: *Pipeline for TT-based global sensitivity analysis, building upon Ch. 6: a model with N input variables is approximated as an N-dimensional tensor, from which we extract a compact 2^N tensor $\mathcal{S}$ approximating all $2^N - 1$ variance components. This tensor can be then used for various aggregation, analysis and query/optimization tasks.*

a significant number N of input variables, remains a major challenge. Even if the chosen surrogate scales well with the dimensionality [Konakli and Sudret, 2016], the sheer number of sensitivity indices is by definition exponential, as there are $2^N - 1$ indices, out of which $\binom{N}{M}$ for any fixed order $1 \leq M \leq N$ may be chosen. For moderate or large values of N, general queries of the form *"retrieve the largest indices of any order"* or *"compute the total variance for interactions of order up to k"* quickly become computationally intractable.

To address these problems we propose a data structure that compactly stores *all* Sobol sensitivity indices in a compressed form, namely in the TT format. While formulas to compute individual or aggregated Sobol indices from various low-rank surrogates have been already derived in the recent literature [Rai, 2014], [Dolgov et al., 2014], [Konakli and Sudret, 2016], our approach is the first to assemble the complete set of indices in a unified and compact tensor format that can be manipulated and queried for statistics, model reduction, visual analytics, and more. See Fig. 7.1 for a summary diagram of our pipeline.

Contribution

We introduce the Sobol tensor $\mathcal{S}$, an N-dimensional TT-compressed multiarray encoding all possible $2^N - 1$ variance components for global SA, and show its derivation from an arbitrary TT surrogate model. We further extract the related aggregated tensors $\mathcal{S}^S$, $\mathcal{S}^C$ and $\mathcal{S}^T$ containing the *superset*, *closed* and *total* Sobol indices, respectively. All these indices can be derived from each other via union/intersection operations that are translated to the tensor-compressed domain as simple matrix additions and subtractions. By combining these ideas with existing

optimization algorithms for the TT format we are able to answer several computationally challenging types of global SA queries that often arise during variable selection and model interpretation.

7.2 Background

7.2.1 Sobol Decomposition

Let us first introduce the ingredients that build up the Sobol indices and their variants. Variance-based SA dates back to the early 20th century and comprises a set of related techniques for statistical analysis of multidimensional data, out of which the ANOVA is arguably the most widely known. Let $F(\mathbf{x}) = F_1(x_1) \cdots F_N(x_N)$ be a separable probability distribution function (PDF) on a rectangle $\Omega = [0, 1]^N$ and $f : \Omega \to \mathbb{R}$ be an L^2-integrable function. Thus, $\mathbf{x} = (x_1, \ldots, x_N) \in \Omega$ is a vector of random variables, distributed according to F, and $f(\mathbf{x})$ has a variance. The goal is to identify how much of that variance is attributable to each individual random variable x_n, and combinations thereof. The *Sobol decomposition*, also known as the functional ANOVA or Hoeffding decomposition [Hoeffding, 1948; Efron and Stein, 1981; Sobol, 1990], splits up f into 2^N terms, each of which depends on a different subset of its input variables:

$$f(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \subseteq \{1,\ldots,N\}} f_{\boldsymbol{\alpha}}(\mathbf{x}) \tag{7.1}$$

where each $f_{\boldsymbol{\alpha}}(\mathbf{x})$ only depends effectively on $\mathbf{x}_{\boldsymbol{\alpha}}$ and is built as

$$\int_{\Omega_{-\boldsymbol{\alpha}}} \left(f(\mathbf{x}) - \sum_{\boldsymbol{\beta} \subsetneq \boldsymbol{\alpha}} f_{\boldsymbol{\beta}}(\mathbf{x}_{\boldsymbol{\beta}}) \right) dF_{-\boldsymbol{\alpha}}(\mathbf{x}_{-\boldsymbol{\alpha}}) \tag{7.2}$$

with $f_\emptyset(\mathbf{x}) = f_\emptyset = \mathbb{E}[f] = \int_\Omega f(\mathbf{x})\, dF(\mathbf{x})$.

7.2.2 Variance Components

The *unnormalized variance components* $D_{\boldsymbol{\alpha}}$ are defined as the variance contributed by each of the $f_{\boldsymbol{\alpha}}$, w.r.t. the PDF F: $D_{\boldsymbol{\alpha}} := \mathrm{Var}[f_{\boldsymbol{\alpha}}]$. Thus, the Sobol decomposition builds up a partition of the overall variance D:

$$D = \sum_{\boldsymbol{\alpha}} D_{\boldsymbol{\alpha}} = \mathrm{V}[f] = \mathrm{E}[f^2] - \mathrm{E}^2[f] = \int_\Omega f(\mathbf{x})^2 dF(\mathbf{x}) - f_\emptyset^2 \tag{7.3}$$

The *normalized variance components* $S_{\boldsymbol{\alpha}}$ (or just *variance components*) in turn map the relative variances w.r.t. the total model variance:

$$\begin{gathered} S : \mathcal{P}(\{1,\ldots,N\}) \setminus \emptyset \to [0,1] \\ S_{\boldsymbol{\alpha}} := D_{\boldsymbol{\alpha}}/D \\ \textstyle\sum_{\boldsymbol{\alpha}} S_{\boldsymbol{\alpha}} = 1 \end{gathered} \tag{7.4}$$

where $\mathcal{P}(\cdot)$ is the *power set* operator, i.e. every non-empty subset of $\{1,\ldots,N\}$ has an associated index.

These indices are an invaluable tool in many SA settings [Saltelli et al., 2004], for example in factor prioritization (reducing uncertainty), factor fixing (identifying non-influential variables), risk minimization, reliability engineering, etc. They can be helpful to select good dimension orderings that lead to more compact surrogate models (example 5.8 from [Bigoni, 2015]; also considered in [Dolgov et al., 2014]). They are hyperedges of a hypergraph, since they encode n-ary relations within subsets of $\{1,\ldots,N\}$. Alternatively they can be thought of in terms of set cardinalities, as the sum of all $S_{\boldsymbol{\alpha}}$ equals 1 (see e.g. [Owen, 2013] and [Saltelli et al., 2008], Sec. 1.2.15).

Several surrogate models lend themselves well to direct estimation of such indices. Examples in the literature include PCE of bounded degree [Sudret, 2008], low-rank sums of separable PCE terms [Konakli and Sudret, 2016], Gaussian processes [Marrel et al., 2009], TT [Dolgov et al., 2014], spectral TT [Bigoni et al., 2016], etc. However, there are $2^N - 1$ possible QOI after excluding the trivial tuple $\boldsymbol{\alpha} = [0,\ldots,0] \equiv \emptyset$. As N grows, this magnitude poses challenges in both the computational and the model interpretation aspects.

7.2.3 Related Indices

One may derive alternative useful indices by adding and/or subtracting together the $S_{\boldsymbol{\alpha}}$, effectively configuring a set algebra.

Total Indices

Denoted as $S^T_{\boldsymbol{\alpha}}$, they are also called *upper indices* [Owen, 2013]. They represent all joint indices that include any variable from $\boldsymbol{\alpha}$:

$$S^T_{\boldsymbol{\alpha}} := \sum_{\boldsymbol{\beta} | \boldsymbol{\alpha} \cap \boldsymbol{\beta} \neq \emptyset} S_{\boldsymbol{\beta}} \tag{7.5}$$

For example, in a 3-variable model we have $S^T_{1,2} = S_1 + S_2 + S_{1,2} + S_{1,3} + S_{2,3} + S_{1,2,3}$. If $|\boldsymbol{\alpha}| = 1$ we are encoding the total influence of a single variable also accounting for its higher-order interactions with all other variables. In this case the indices sometimes are called *total effects* [Homma and Saltelli, 1996], and have been used to identify and select the most relevant variables, for example by sorting S^T_n and choosing the k largest [Fock, 2014], [Abualrub et al., 2017].

However, this criterion may lead to overestimating variables that exhibit large overlapping variance contributions.

Closed Indices

Denoted as $S^C_{\boldsymbol{\alpha}}$, they are also called *first-order indices* [Sudret, 2008] or *lower indices* [Owen, 2013]. They sum the variance contributions of all non-empty tuples contained in $\boldsymbol{\alpha}$:

$$S^C_{\boldsymbol{\alpha}} := \sum_{\boldsymbol{\beta} | \boldsymbol{\alpha} \supseteq \boldsymbol{\beta}} S_{\boldsymbol{\beta}} \tag{7.6}$$

For example, for 3 variables we have $S^C_{1,2} = S_1 + S_2 + S_{1,2}$. Also, for any single variable n we have $S^C_n = S_n$. The closed indices can be written in terms of the total indices as $S^T_{\boldsymbol{\alpha}} = 1 - S^T_{-\boldsymbol{\alpha}}$.

Superset Indices

The $\mathcal{S}^S$ aggregate all indices that contain a tuple [Hooker, 2004]:

$$S^S_{\boldsymbol{\alpha}} := \sum_{\boldsymbol{\beta} | \boldsymbol{\alpha} \subseteq \boldsymbol{\beta}} S_{\boldsymbol{\beta}} \tag{7.7}$$

For example, $S^S_{1,2} = S_{1,2} + S_{1,2,3}$.

7.2.4 Tensor Surrogates and Sensitivity Analysis

Tensor decompositions make for attractive surrogates owing to their natural multidimensionality and fast decompression, as also argued in the previous chapter. Several examples [Espig et al., 2011], [Litvinenko et al., 2013], [Dolgov et al., 2014] target solutions of multiparametric partial differential equations (PDEs). [Konakli and Sudret, 2016] proposed an interpolator via sums of separable PCE-based functions (low-rank approximations, LRA) and showed how to extract Sobol indices out of them. This is related to the CP decomposition, with the main difference that their factors are continuous and are sought within the subspace spanned by a few leading orthogonal polynomials. [Vervliet et al., 2014] demonstrated CP-based tensor completion and visualization for the melting point of an alloy, depending on the concentration of its 10 different constituent materials. [Ballester-Ripoll et al., 2016] proposed visualization diagrams for TT-format surrogates of several mechanical simulations, emphasizing multidimensionality and real-time reconstruction.

A few papers have extracted Sobol indices from TT surrogates. [Dolgov et al., 2014] build their decomposition using ACA and derive properties and statistics

including means, covariances, level sets, and individual Sobol indices. [Zhang et al., 2015b] developed a hierarchical uncertainty quantification algorithm using TT and PCE to estimate a circuit's response depending on its subcomponents' behavior. [Rai, 2014] gives formulas to compute Sobol indices from a range of low-rank approximation surrogates, including TT-based.

7.3 The Sobol Tensor Train

We now introduce our proposed *Sobol tensor train*, denoted as $\mathcal{S}$, which has dimension N and size 2 along each dimension for a total of 2^N elements. Such $2 \times \cdots \times 2$ tensors are not unusual, see for example the so-called *quantized tensor train* (QTT) and the closely-related *wavelet tensor train* (WTT) [Oseledets and Tyrtyshnikov, 2011], as well as the recent multilinear regressors known as *exponential machines* [Novikov et al., 2016]. $\mathcal{S}$ hence records the variance components for all n-ary interactions:

$$S_{\boldsymbol{\alpha}} \approx \mathcal{S}_{\boldsymbol{\alpha}} = \mathcal{S}[j_1, \ldots, j_N] = \mathcal{S}^{(1)}[j_1] \cdot \cdots \cdot \mathcal{S}^{(N)}[j_N] \tag{7.8}$$

with $j_n = 1$ if $n \in \boldsymbol{\alpha}$ and 0 otherwise (recall that we are using the shortcut $[k]$ to indicate the k-th slice along the second mode, i.e. $[:, k, :]$). To construct it we combine the definitions of Sobol decomposition (Sec. 7.2.1) and variance components (Sec. 7.2.2) with the TT formulation as follows.

Proposition 7.3.1. *Let $\mathbf{x} = (x_1, \ldots, x_N)$ be a vector of independent random variables with distributions $F_1, \ldots, F_N$, and let $\mathcal{T} = [[\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N)}]]$ be a TT surrogate $\tilde{f}(\mathbf{x}) \approx f(\mathbf{x})$. Then, each term $\tilde{f}_{\boldsymbol{\alpha}}$ of the Sobol decomposition of $\tilde{f}$ is given by $\mathcal{T}_{\boldsymbol{\alpha}} = [[\mathcal{T}_{\boldsymbol{\alpha}}^{(1)}, \ldots, \mathcal{T}_{\boldsymbol{\alpha}}^{(N)}]]$, where each core slice (matrix) is defined as*

$$\mathcal{T}_{\boldsymbol{\alpha}}^{(n)}[i_n] := \begin{cases} \mathrm{E}[\mathcal{T}^{(n)}] \text{ if } n \notin \boldsymbol{\alpha} \\ \mathcal{T}^{(n)}[i_n] - \mathrm{E}[\mathcal{T}^{(n)}] \text{ if } n \in \boldsymbol{\alpha} \end{cases} \tag{7.9}$$

for all n and all slices $i_n = 0, \ldots, I_n - 1$, where $\mathrm{E}[\mathcal{T}^{(n)}] := \frac{1}{I_n} \cdot \sum_{i=0}^{I_n - 1} F_n[i_n]\, \mathcal{T}^{(n)}[i_n]$ is the discretized expectation operator along the n-th dimension, i.e. a matrix obtained as the average of the n-th core's slices, weighted by the n-th PDF term.

Proof. Consider a tuple $\boldsymbol{\alpha}$ and an arbitrary sampling point $\mathbf{x} = (x_1, \ldots, x_N)$ corresponding to the tensor entry $\mathbf{i} = (i_1, \ldots, i_N)$. We want to show that the TT approximation of $\tilde{f}_{\boldsymbol{\alpha}}$ is

$$\mathcal{T}_{\boldsymbol{\alpha}}(\mathbf{x}) = \prod_{n=1}^{N} \mathcal{T}_{\boldsymbol{\alpha}}^{(n)}[i_n] \tag{7.10}$$

with $\mathcal{T}^{(n)}_{\boldsymbol{\alpha}}[i_n]$ as in Eq. 7.9. Expanding the $|\boldsymbol{\alpha}|$ subtractions that are multiplied together in Eq. 7.10 we get a sequence of $2^{|\boldsymbol{\alpha}|}$ additions and subtractions:

$$\sum_{\boldsymbol{\beta}|\boldsymbol{\beta}\subseteq\boldsymbol{\alpha}} (-1)^{|\boldsymbol{\alpha}|-|\boldsymbol{\beta}|} \prod_{n=1}^{N} \widehat{\mathcal{T}}^{(n)}_{\boldsymbol{\beta}}[i_n] \tag{7.11}$$

with

$$\widehat{\mathcal{T}}^{(n)}_{\boldsymbol{\beta}}[i_n] := \begin{cases} \mathrm{E}[\mathcal{T}^{(n)}] \text{ if } n \notin \boldsymbol{\beta} \\ \mathcal{T}^{(n)}[i_n] \text{ if } n \in \boldsymbol{\beta} \end{cases} \tag{7.12}$$

Recall that $\mathcal{T}^{(n)}$ encodes the model $\tilde{f}$'s response along the n-th axis, while $\mathrm{E}[\mathcal{T}^{(n)}]$ represents its integration along that axis. Therefore Eq. 7.11 becomes

$$\begin{aligned} &\sum_{\boldsymbol{\beta}\subseteq\boldsymbol{\alpha}} (-1)^{|\boldsymbol{\alpha}|-|\boldsymbol{\beta}|} \int_{\Omega-\boldsymbol{\beta}} \tilde{f}(\mathbf{x}) dF_{-\boldsymbol{\alpha}}(\mathbf{x}_{-\boldsymbol{\alpha}}) \\ = \int_{\Omega-\boldsymbol{\alpha}} \tilde{f}(\mathbf{x})\, dF_{-\boldsymbol{\alpha}}(\mathbf{x}_{-\boldsymbol{\alpha}}) &- \sum_{\boldsymbol{\beta}|\boldsymbol{\beta}\subset\boldsymbol{\alpha}} \tilde{f}_{\boldsymbol{\beta}}(\mathbf{x}_{\boldsymbol{\beta}}) = \tilde{f}_{\boldsymbol{\alpha}}(\mathbf{x}_{\boldsymbol{\alpha}})\,\square \end{aligned} \tag{7.13}$$

Eq. 7.9 can be intuitively interpreted as follows: Variables not in $\boldsymbol{\alpha}$ must be integrated over their domain of existence, and $\tilde{f}_{\boldsymbol{\alpha}}$ does not effectively depend on them. Their corresponding cores are accordingly constant. For variables in $\boldsymbol{\alpha}$, on the other hand, we must keep the original function but subtract from it the lower-order expectations; these are all correctly accounted for thanks to multilinearity.

Since we can obtain the term $f_{\boldsymbol{\alpha}}$ for any $\boldsymbol{\alpha}$ via Eq. 7.9, we can already obtain any arbitrary variance component $S_{\boldsymbol{\alpha}}$ by computing the variance of $\mathcal{T}_{\boldsymbol{\alpha}}$. However, a more expeditious method allows us to produce *all* components at once. First note that the following tensor $\mathcal{T}_*$ of size $(I_1+1)\times\cdots\times(I_N+1)$ compactly encodes all 2^N Sobol decomposition terms:

$$\begin{cases} \mathcal{T}^{(n)}_*[0] := \mathrm{E}[\mathcal{T}^{(n)}] \\ \mathcal{T}^{(n)}_*[j] := \mathcal{T}^{(n)}[j-1] - \mathrm{E}[\mathcal{T}^{(n)}] \text{ for } j = 1,\ldots,I_n \end{cases} \tag{7.14}$$

Each core in the tensor $\mathcal{T}_*$ is but a combination of the two types of slices of Eq. 7.9. Therefore it contains $\mathcal{T}_\alpha$ for *all* possible $\boldsymbol{\alpha} \subseteq \{1,\ldots,N\}$, and so it approximates $f_{\boldsymbol{\alpha}}(\mathbf{x})\,\forall\boldsymbol{\alpha},\mathbf{x}$. We have now all necessary components to construct our Sobol tensor $\mathcal{S}$: we need to compute $\mathrm{V}[f_{\boldsymbol{\alpha}}]/D = \mathrm{E}[(f_{\boldsymbol{\alpha}} - \mathrm{E}[f_{\boldsymbol{\alpha}}])^2]/D$ in the compressed domain. The procedure, detailed in Alg. 9, also obtains the unnormalized variance indices $D_{\boldsymbol{\alpha}}$.

If the input surrogate has TT-ranks $R_1,\ldots,R_{N-1}$, then $\mathcal{S}$ may have at most ranks $R_1^2,\ldots,R_{N-1}^2$. The squaring (line 2 from Alg. 9) can be achieved either

Algorithm 9 Given a TT surrogate $\mathcal{T} = [[\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(N)}]]$ of size $I_1 \times \cdots \times I_N$, extract the compressed Sobol tensor $\mathcal{S}$ of size $2 \times \cdots \times 2$.

Compute $\mathcal{T}_*$ as in Eq. 7.14 $\{\mathcal{T}_*$ encodes $f_{\boldsymbol{\alpha}}\, \forall \boldsymbol{\alpha}\}$
Compute $\mathcal{T}_{**} := \mathcal{T}_* \circ \mathcal{T}_* = \mathcal{T}_*^2$ $\{\mathcal{T}_{**}$ encodes $f_{\boldsymbol{\alpha}}^2\, \forall \boldsymbol{\alpha}\}$
for $n = 1, \dots, N$ **do**
    $\mathcal{D}^{(n)}[0] := \mathcal{T}_{**}^{(n)}[0]$
    $\mathcal{D}^{(n)}[1] := \frac{1}{I_n} \cdot \sum_{i=0}^{I_n - 1} F_n(x_n(i))\, \mathcal{T}_{**}^{(n)}[i+1]$
end for
$\mathcal{D} := [[\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(N)}]]$ $\{\mathcal{D}$ encodes $\mathrm{V}[f_{\boldsymbol{\alpha}}]\, \forall \boldsymbol{\alpha}\}$
$D := \prod_{n=1}^{N} (\mathcal{D}^{(n)}[0] + \mathcal{D}^{(n)}[1])$ {Total variance $\mathrm{V}[f]$}
$\mathcal{S} := \mathcal{D}/D$ {Normalize variances by overall variance D}
return $\mathcal{S}$

by ACA or by slice-wise Kronecker product if the rank is low enough (recall Sec. 2.7.2). All other operations cannot increase any of the ranks. Lastly, note that the first coefficient in the tensor $\mathcal{S}_\emptyset = \mathcal{S}[0, \dots, 0] = \tilde{f}_\emptyset / D$ is not a Sobol index; we set it to zero if needed with a simple rank-1 correction:

$$\mathcal{S} \leftarrow \mathcal{S} - \begin{pmatrix} \tilde{f}_\emptyset / D \\ 0 \end{pmatrix} \otimes \overbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix}}^{N-1 \text{ terms}} \tag{7.15}$$

7.4 Computing Aggregated Indices

Aggregated indices require up to an exponential number of addends if computed naively. But thanks to the multilinearity of the proposed tensor decomposition, we can obtain all such QOI at once and at very little cost as we describe next.

Superset Sobol Tensors

We recall now the notion of superset indices from Sec. 7.2.3, which capture the aggregate dependence with respect to a group of indices; i.e. variance components over $\boldsymbol{\alpha}$ plus the indices of all variable tuples that are a superset of $\boldsymbol{\alpha}$. If $\mathcal{S}$ is available, we can derive a superset Sobol tensor $\mathcal{S}^S = [[\mathcal{S}^{S(1)}, \dots, \mathcal{S}^{S(N)}]]$ that approximates any $S_{\boldsymbol{\alpha}}^S \approx \mathcal{S}_{\boldsymbol{\alpha}}^S$. We construct its cores by slice-wise manipulation of the original cores:

$$\begin{cases} \mathcal{S}^{S(n)}[0] := \mathcal{S}^{(n)}[0] + \mathcal{S}^{(n)}[1] \\ \mathcal{S}^{S(n)}[1] := \mathcal{S}^{(n)}[1] \end{cases} \tag{7.16}$$

The rationale is that variables that are present in a tuple (encoded by the second slices, $j = 1$) should stay present, while the rest (first slices, $j = 0$) should be accounted for both when they are absent and when they are included. As an example, let us consider in 2D the second superset index $\mathcal{S}_2^S := \mathcal{S}_{12} + \mathcal{S}_2$. Eq. 7.16 yields

$$\begin{aligned} \mathcal{S}_2^S &= \mathcal{S}^{S(1)}[0] \cdot \mathcal{S}^{S(2)}[1] = (\mathcal{S}^{(1)}[0] + \mathcal{S}^{(1)}[1]) \cdot \mathcal{S}^{(2)}[1] \\ &= \mathcal{S}^{(1)}[0] \cdot \mathcal{S}^{(2)}[1] + \mathcal{S}^{(1)}[1] \cdot \mathcal{S}^{(2)}[1] = \mathcal{S}_1 + \mathcal{S}_{12} \end{aligned} \tag{7.17}$$

as expected. Conversely, one may extract $\mathcal{S}$ from $\mathcal{S}^S$ by reverting the slice-wise transformations:

$$\begin{cases} \mathcal{S}^{(n)}[0] = \mathcal{S}^{S(n)}[0] - \mathcal{S}^{S(n)}[1] \\ \mathcal{S}^{(n)}[1] = \mathcal{S}^{S(n)}[1] \end{cases} \tag{7.18}$$

We wish to emphasize the compactness and convenience of the relations given by Eqs. 7.16 and 7.18. A naive sum to obtain a superset index of order K out of the variance components $\mathcal{S}$ would require 2^{N-K} additions. For example, for $N = 3$ and $\boldsymbol{\alpha} = \{1\}$ we have $\mathcal{S}_1^S = \mathcal{S}_1 + \mathcal{S}_{12} + \mathcal{S}_{13} + \mathcal{S}_{123}$. Conversely, producing indices $\mathcal{S}$ from $\mathcal{S}^S$ needs 2^{N-K} mixed additions and subtractions as dictated by the inclusion-exclusion principle from combinatorics. For instance, $\mathcal{S}_1 = \mathcal{S}_1^S - \mathcal{S}_{12}^S - \mathcal{S}_{13}^S + \mathcal{S}_{123}^S$. On the other hand, Eqs. 7.16 and 7.18 need only $O(NR^2)$ operations in the TT format.

Closed Sobol Tensors

Similarly to Eq. 7.16, we derive the closed Sobol tensor $\mathcal{S}^C$ from $\mathcal{S}$ as follows:

$$\begin{cases} \mathcal{S}^{C(n)}[0] := \mathcal{S}^{(n)}[0] \\ \mathcal{S}^{C(n)}[1] := \mathcal{S}^{(n)}[0] + \mathcal{S}^{(n)}[1] \end{cases} \tag{7.19}$$

The logic here is that indices absent in a tuple should stay absent, while present indices should be accounted for when they are missing also (since we want to sum all subsets). The converse equation reads

$$\begin{cases} \mathcal{S}^{(n)}[0] = \mathcal{S}^{C(n)}[0] \\ \mathcal{S}^{(n)}[1] = \mathcal{S}^{C(n)}[1] - \mathcal{S}^{C(n)}[0] \end{cases} \tag{7.20}$$

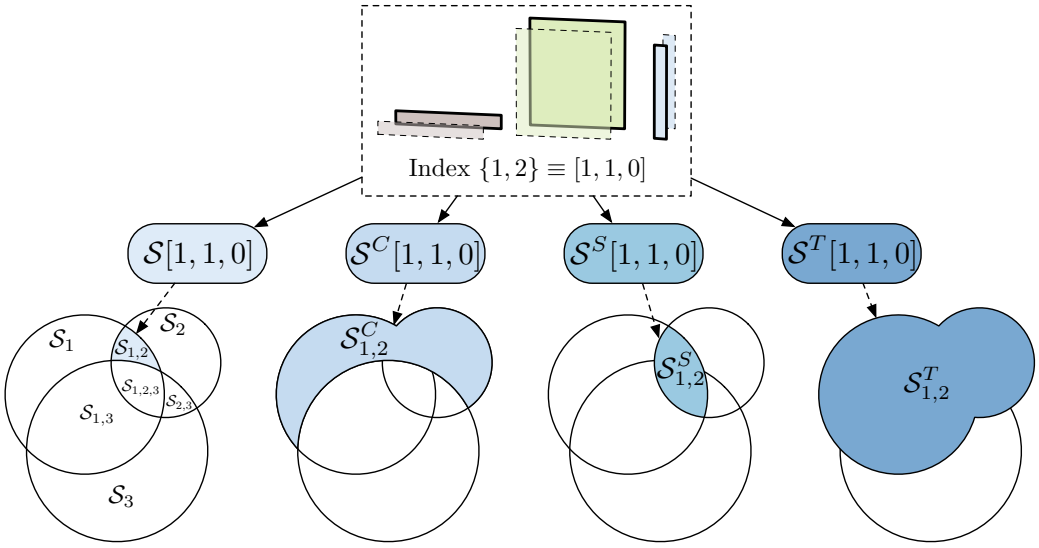


Figure 7.2: *Examples of variance components $\mathcal{S}$ and total $\mathcal{S}^T$, superset $\mathcal{S}^S$, and closed Sobol indices $\mathcal{S}^C$ for a 3-variable model, interpreted as set cardinalities. Each colored region area is obtained by multiplying the indexed slices from its corresponding tensor.*

Total Sobol Tensors

Our last aggregated tensor is the total $\mathcal{S}^T$ and can be obtained via the complement operation as $\mathcal{S}^T_{\boldsymbol{\alpha}} = 1 - \mathcal{S}^C_{-\boldsymbol{\alpha}}$. Let us define a complement tensor $\bar{\mathcal{S}}^C$, defined for each tuple as $\bar{\mathcal{S}}^C_{\boldsymbol{\alpha}} := \mathcal{S}^C_{-\boldsymbol{\alpha}}$. We extract this tensor from $\mathcal{S}^C$ by simply swapping the two slices of each core:

$$\begin{cases} \bar{\mathcal{S}}^{C(n)}[0] := \mathcal{S}^{C(n)}[1] \\ \bar{\mathcal{S}}^{C(n)}[1] := \mathcal{S}^{C(n)}[0] \end{cases} \tag{7.21}$$

and the final result $\mathcal{S}^T = 1 - \bar{\mathcal{S}}^C$ follows from a tensor-tensor element-wise subtraction (accomplished as described in Sec. 2.7.2). To retrieve $\mathcal{S}^C$ back from $\mathcal{S}^T$ it suffices to repeat the whole transformation.

7.5 Global Sensitivity Metrics and Queries

7.5.1 Relevant Subsets of Variables

A classical task in SA is to "*select the k variables that account for the most variance*", or alternatively "*select the smallest set variables that account for at least (say) 99% variance*". The mask tensor $\mathcal{M}^k$, introduced back in Ch. 6, allows

us to define restricted searches. For instance, the single most important variance component of order k is

$$\arg\max_{\boldsymbol{\alpha}} \left\{ (\mathcal{S} \circ \mathcal{M}^k)_{\boldsymbol{\alpha}} \right\} \tag{7.22}$$

which we solve using a state-of-the-art global optimization algorithm in the TT format, just as before. One may also use $\mathcal{S}^T, \mathcal{S}^C$ or $\mathcal{S}^S$ instead of $\mathcal{S}$ depending on the task at hand. For example, the $\mathcal{S}_{\boldsymbol{\alpha}}$ do play the dominant role in factor prioritization, but for factor fixing one is advised to seek a tuple with the smallest total index [Saltelli et al., 2008].

We also use $\mathcal{M}^k$ to compute the overall per-order contributions: the tensor dot product

$$< \mathcal{S}, \mathcal{M}^k > \tag{7.23}$$

gives us the combined order k indices $\sum_{|\boldsymbol{\alpha}|=k} \mathcal{S}_{\boldsymbol{\alpha}}$.

7.5.2 Other Constraints

The mask tensor $\mathcal{M}^k$ restricts optimization queries such as Eq. 7.22 to tuples of a given order. However, the analyst may seek a model simplification that satisfies different (and possibly additional) constraints, e.g. that certain variables must, or must not, become frozen. Such conditions can be easily encoded on a new mask tensor $\mathcal{M}$. This mask can be, say, filled with 1's, or can be a Hamming mask $\mathcal{M}^k$. Then, if for instance a variable $1 \leq n \leq N$ should be fixed (i.e. simplified) it is sufficient to fill the second slice of the n-th core of $\mathcal{M}$ with zeros. This effectively restricts the search to $\{\boldsymbol{\alpha} \,|\, n \notin \boldsymbol{\alpha}\}$ as desired, because all entries $\mathcal{M}_{\boldsymbol{\alpha}}$ where $n \in \boldsymbol{\alpha}$ will become zero. Conversely, if we wish to ensure that a variable is not fixed (i.e. remains active in the new simplified model), we just have to fill the *first* slice of the n-th core of $\mathcal{M}$ with zeros.

7.6 Results

Our experiments were conducted in Python 3.5 using the same environment as the previous chapter.

7.6.1 Sobol "G" Function

This function has been extensively used in the SA literature owing to its flexibility and relatively high-order interactions. It is defined as

$$f(\mathbf{x}) := \prod_{n=1}^{N} \frac{|4x_n - 2| + a_n}{1 + a_n} \tag{7.24}$$

being a_i random coefficients sampled from a uniform distribution $\mathcal{U}(0,1)$ and $x_n \sim \mathcal{U}(0,1)$ the n function variables. Note that f is non-differentiable at one point, namely $(0.5,\ldots,0.5)$. We can expect to get an exact TT interpolator (up to machine precision) of f as it is a product of univariate functions and therefore it has a rank 1. Our test example uses $N = 25$ dimensions and we discretize each variable into $I_1 = \cdots = I_{25} = 64$ possible values. The ACA used 3200 evaluations of f and was able to achieve a relative error of $\epsilon \approx 4.646 \cdot 10^{-15}$ over a test set of 4096 samples drawn at random using LHS. Extracting the Sobol TT from the surrogate took 4.93 seconds using $\epsilon = 10^{-6}$ as the ACA relative error for the squaring step in Alg. 9. Fig. 7.3 shows every value of a_n and its corresponding first-order Sobol index, computed using our method (Alg. 9).

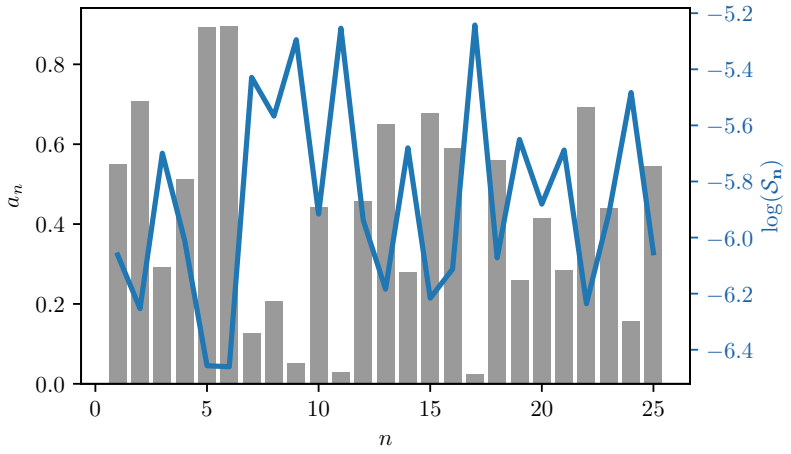


Figure 7.3: *The 25 random values chosen for a_n and their resulting Sobol values (depicted in logarithmic scale).*

Tab. 7.1 shows the 5 highest variance components of any order from our method, which in this case are only order-1 effects. We also show the indices as estimated directly from sampling the TT surrogate via the Sensitivity Analysis Library (SALib [Herman and Usher, 2017]) in Python, with varying number of sample points P. Finally and to complete the cross-check, we list the analytical Sobol values for comparison [Sobol, 2003]: $D_n = 1/(3(1+a_n)^2)$, $D = \prod_n (D_n+1) - 1$, and $S_n = (\prod_n D_n)/D$. Tab. 7.2 shows the highest aggregated indices (i.e. total, closed and superset) of order 1, 2, and 3 separately.

We observe that the TT indices are accurate to almost 4 decimal digits. The indices computed by SALib become closer as more samples are taken, further supporting the correctness of our method. Note that for this function SALib required a very large number of samples to obtain results with a similar level of

Table 7.1: *Highest variance components for the Sobol G function*

Index	Value				
	Sobol TT	SALib			Analytical
		$(P = 520000)$	$(P = 5.2 \cdot 10^6)$	$(P = 5.2 \cdot 10^7)$	
$\mathcal{S}_{17}$	0.0053	0.0147	0.0110	0.0073	0.0054
$\mathcal{S}_{11}$	0.0052	-0.0001	0.0053	0.0052	0.0054
$\mathcal{S}_{9}$	0.0050	0.0237	0.0084	0.0053	0.0051
$\mathcal{S}_{7}$	0.0044	-0.0038	0.0035	0.0039	0.0045
$\mathcal{S}_{24}$	0.0042	0.0259	0.0062	0.0042	0.0042

Table 7.2: *Highest aggregated indices of order 1, 2, and 3 for the Sobol G function*

Order	Index / Variable(s)		
	Total	Closed	Superset
1	$\mathcal{S}^T_{17} = 0.2452$	$\mathcal{S}^C_{17} = 0.0053$	$\mathcal{S}^S_{17} = 0.2452$
2	$\mathcal{S}^T_{11,17} = 0.4296$	$\mathcal{S}^C_{11,17} = 0.0122$	$\mathcal{S}^S_{11,17} = 0.0586$
3	$\mathcal{S}^T_{9,11,17} = 0.5657$	$\mathcal{S}^C_{9,11,17} = 0.0209$	$\mathcal{S}^S_{9,11,17} = 0.0136$

precision as compared to our proposed method. We attribute this to the function's high dimensionality, which can be nonetheless well handled with the TT.

7.6.2 Piston Simulation

This is a more complex model that measures the cycle time of a piston simulation [Kenett and Zacks, 1998]. The output is defined analytically on 7 variables as

$$f(\mathbf{x}) = 2\pi\sqrt{\frac{M}{k + S^2\frac{P_0V_0}{T_0}\frac{T_a}{V^2}}} \tag{7.25}$$

with

$$\begin{aligned} V &= \frac{S}{2k}\left(\sqrt{A^2 + 4k\frac{P_0V_0}{T_0}T_a} - A\right) \\ A &= P_0S + 19.62M - \frac{kV_0}{S} \end{aligned} \tag{7.26}$$

The full list of parameters and their input ranges is detailed in Tab. 7.3. Our model was generated with ACA, stopped after 43904 function evaluations, again

Table 7.3: *Parameters of the Piston function*

Variable	Description	Units	Distribution
M	Piston weight	kg	$\mathcal{U}(30, 60)$
S	Piston surface area	m^2	$\mathcal{U}(0.005, 0.02)$
V_0	Initial gas volume	m^3	$\mathcal{U}(0.002, 0.01)$
k	Spring coefficient	N/n	$\mathcal{U}(1000, 5000)$
P_0	Atmospheric pressure	N/m^2	$\mathcal{U}(90000, 110000)$
T_a	Ambient temperature	K	$\mathcal{U}(290, 296)$
T_0	Filling gas temperature	K	$\mathcal{U}(340, 360)$

with $I = 64$ bins per dimension. It has 10496 non-zero elements and maximum rank $R = 7$, and it achieves $\epsilon \approx 0.077\%$ over an LHS-acquired test set. Note that the TT model is again built with fewer samples than those needed by SALib's MC algorithm, and that it is able to compute indices of arbitrary order *a posteriori*. Extracting the Sobol TT took 5.70 seconds in this case.

For further comparison we have also computed a PCE approximation of this function via 4096 training samples chosen similarly to the test set. To build the model we take the 4 first Legendre polynomials for each variable. Then we compress the PCE-Tucker core into a TT model as detailed in Sec. 6.3.5 with a relative error of 0.5%, resulting in $R = 22$. The resulting TT-PCE model approximates the training set with a relative error $\epsilon \approx 0.38\%$, and achieves $\epsilon \approx 1.22\%$ on the test set.

As shown in Tab. 7.4, our analysis reveals that only the 4 first variables have a significant first-order effect. Their numerical values are consistent with the results reported in [Owen et al., 2013] (after normalization). Also, the most important tuple interactions arise from these very same variables. The triplet $\{S, V_0, k\}$ in particular has a closed index of about 95% as reported in Tab. 7.5. Overall, interactions of order 3 and above play a relatively small role.

7.6.3 GEMM Product

In this final experiment we compute the Sobol indices (Tabs 7.6 and 7.7) out of the TT model learned in the previous chapter (Sec 6.6.3). The Sobol tensor took 12.32 seconds to build. Our results indicate a relatively large presence of high-order interactions; this matches the prior knowledge that GPU kernel optimization is a challenging high-dimensional parameter space, and that the parameters' influences tend to be highly inter-dependent [Nugteren and Codreanu, 2015]. In particular the most important first-order index (from M_{wg}) is only about 6%, and all order-1 indices combined explain only less than one fourth of the total model

Table 7.4: *Highest variance components for the piston function (interactions of order 3 and above are not supported by SALib)*

Index	Var(s)	Value			
		Sobol TT	Sobol TT-PCE	SALib on TT (P=160000)	SALib (P=160000)
$\mathcal{S}_2$	S	0.5545	0.5585	0.5562	0.5563
$\mathcal{S}_3$	V_0	0.3207	0.3238	0.3215	0.3215
$\mathcal{S}_1$	M	0.0390	0.0396	0.0389	0.0391
$\mathcal{S}_{2,4}$	S,k	0.0242	0.0211	0.0252	0.0250
$\mathcal{S}_4$	k	0.0212	0.0200	0.0219	0.0221
$\mathcal{S}_{3,4}$	V_0,k	0.0129	0.0117	0.0121	0.0118
$\mathcal{S}_{2,3,4}$	S,V_0,k	0.0094	0.0066	-	-
$\mathcal{S}_{1,3}$	M,V_0	0.0050	0.0046	0.0053	0.0053
$\mathcal{S}_{2,3}$	S,V_0	0.0046	0.0043	0.0045	0.0044
$\mathcal{S}_{1,2}$	M,S	0.0046	0.0048	0.0036	0.0035

Table 7.5: *Highest aggregated indices of order 1, 2, and 3 for the piston function*

Order	Index / Variable(s)		
	Total	Closed	Superset
1	$\mathcal{S}_2^T = 0.5987$ $\{S\}$	$\mathcal{S}_2^C = 0.5545$ $\{S\}$	$\mathcal{S}_2^S = 0.5987$ $\{S\}$
2	$\mathcal{S}_{2,3}^T = 0.9374$ $\{S,V_0\}$	$\mathcal{S}_{2,3}^C = 0.8799$ $\{S,V_0\}$	$\mathcal{S}_{2,4}^S = 0.0343$ $\{S,k\}$
3	$\mathcal{S}_{1,2,3}^T = 0.9776$ $\{M,S,V_0\}$	$\mathcal{S}_{2,3,4}^C = 0.9475$ $\{S,V_0,k\}$	$\mathcal{S}_{2,3,4}^S = 0.0098$ $\{S,V_0,k\}$

variability. We also use this real-world data set to test our querying routines; we report some sample results in Tab. 7.8 involving various aggregated indices.

To conclude this section we show in Fig. 7.4 one bar chart per data set, containing the overall relative variance broken down by interaction order.

7.7 Discussion

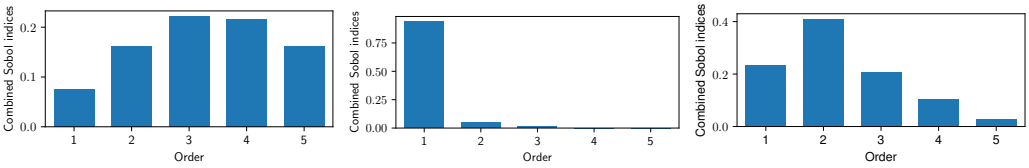
We have introduced a compact data structure that gathers all Sobol indices from any TT-based surrogate model, and have given algorithms to extract various aggregated indices from it. The proposed aggregation algorithms capitalize on the

Table 7.6: *Highest variance components for the GEMM matrix product function*

Index	Var(s)	Value	
		Sobol TT	SALib on TT (P=300000)
$\mathcal{S}_{1,2,4}$	M_{wg},N_{wg},M_{dimC}	0.0842	-
$\mathcal{S}_{1,4}$	M_{wg},M_{dimC}	0.0790	0.0799
$\mathcal{S}_{2,4}$	N_{wg},M_{dimC}	0.0675	0.0754
$\mathcal{S}_{1}$	M_{wg}	0.0643	0.0539
$\mathcal{S}_{1,2}$	M_{wg},N_{wg}	0.0628	0.0686
$\mathcal{S}_{1,4,5}$	M_{wg},M_{dimC},N_{dimC}	0.0330	-
$\mathcal{S}_{1,2,4,5}$	M_{wg},N_{wg},M_{dimC},N_{dimC}	0.0319	-
$\mathcal{S}_{4}$	M_{dimC}	0.0286	0.0257
$\mathcal{S}_{4,5}$	M_{dimC},N_{dimC}	0.0225	0.0165
$\mathcal{S}_{2}$	N_{wg}	0.0198	0.0051

Table 7.7: *Highest aggregated indices of order 1, 2, and 3 for the GEMM matrix product*

Order	Index / Variable(s)		
	Total	Closed	Superset
1	$\mathcal{S}_1^T = 0.6979$ $\{M_{wg}\}$	$\mathcal{S}_1^C = 0.0643$ $\{M_{wg}\}$	$\mathcal{S}_1^S = 0.6979$ $\{M_{wg}\}$
2	$\mathcal{S}_{1,4}^T = 0.8960$ $\{M_{wg},M_{dimC}\}$	$\mathcal{S}_{1,4}^C = 0.1718$ $\{M_{wg},M_{dimC}\}$	$\mathcal{S}_{1,4}^S = 0.4380$ $\{M_{wg},M_{dimC}\}$
3	$\mathcal{S}_{1,2,4}^T = 0.9540$ $\{M_{wg},N_{wg},M_{dimC}\}$	$\mathcal{S}_{1,2,4}^C = 0.4060$ $\{M_{wg},N_{wg},M_{dimC}\}$	$\mathcal{S}_{1,2,4}^S = 0.2301$ $\{M_{wg},N_{wg},M_{dimC}\}$



(a) Sobol G function (b) Piston (c) GEMM

Figure 7.4: *Combined contributions for orders 1 to 5 for all three models, efficiently computed using Eq. 7.23*

format's multilinearity and have very little overhead cost. We combine these ideas with mask tensors, which allow us to define restricted queries and thus aid in model reduction/interpretation tasks. We believe the tensor train has a great potential as a canonical format for approximation of multiparametric systems, and

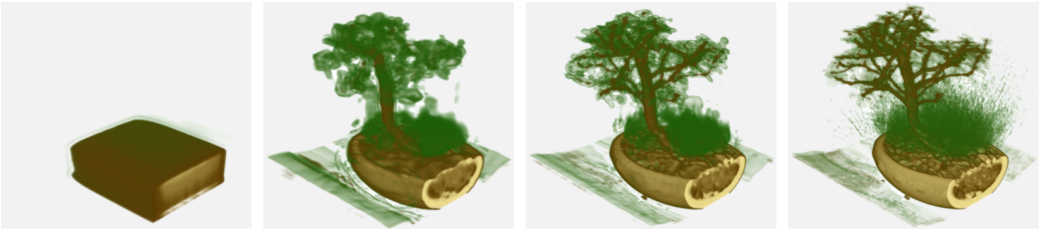
Table 7.8: *Once the Sobol TT is available, we can satisfy efficiently various types of queries as detailed in Sec. 7.5 using mask tensors, constrained search and TT global optimization*

Query	**Result**	**Value**	**Computing time (s)**
Variable that interacts the most with $\{L\$_A, L\$_B\}$	M_{wg}	$\mathcal{S}^S_{1,13,14} = 0.0263$ (as high as possible)	0.3598
Variable that interacts the least with M_{wg}	N_{stride}	$\mathcal{S}^S_{1,12} = 0.0069$ (as low as possible)	0.4237
Highest closed 3-tuple that avoids M_{wg}	N_{wg}, M_{dimC}, N_{dimC}	$\mathcal{S}^C_{2,4,5} = 0.1828$ (as high as possible)	0.2786
Highest closed 3-tuple that includes M_{wg}	M_{wg}, N_{wg}, M_{dimC}	$\mathcal{S}^C_{1,2,4} = 0.5940$ (as high as possible)	0.3686
Six variables that can be frozen with the least impact	K_{wg}, K_{wi}, M_{vec}, $N_{vec}, M_{stride}, N_{stride}$	$\mathcal{S}^T_{3,8,9,10,11,12} = 0.2365$ (as low as possible)	0.7307

the proposed methods for sensitivity analysis can be understood in the context of this trend. The presented framework is flexible in a variety of settings, and supports arbitrary orders of significant variable interactions in higher-dimensional models.

C H A P T E R 8

CONCLUSIONS



8.1 Summary

In this thesis we have contributed several novel tensor-based algorithms to tackle a range of compelling challenges in large-scale signal processing, scientific analysis, and visualization. We believe the proposed pipeline answers satisfactorily RQ1 and RQ2 (Sec. 1), and we hope it will help consolidate the tensor decomposition framework as a powerful toolbox for multidimensional data in these fields.

We first focused on large data sets of relatively low dimensionality that are fully known in an explicit form. The proposed TTHRESH (Ch. 3) strives to optimize Tucker-based compression at the transform-coefficient level. It is the first approach to do so via adaptive quantization, and it achieves excellent compression rates at the error tolerance levels that are adequate for visualization. In Ch. 4 we incorporated compressed-domain filtering operations into an octree multiresolution structure. To this end we employ full-resolution factors together with variable-resolution cores. The former are filtered and downsampled on demand and then used to back-project the corresponding brick cores to ensure a homogeneous response across varying LOD. Our system is a coherent extension to previous tensor-based multiresolution visualization systems. A further application of factor manipulation was given in Ch. 5, where we query compressed summed area tables and integral histograms via matrix row subtraction and column-wise convolution. Our look-up data structure provides a compromise between the high speed of integral histograms and the space efficiency of brute-force traversal.

In the second part of the dissertation we dealt with sparse data, namely observations from experiments and numerical simulations with up to dozens of dimensions. We treated this as a regression problem to be solved in the TT format in Ch. 6, where we also used TT global optimization and outlined and tested several visualization diagrams. The last technical contribution, Ch. 7, builds further upon these ideas and makes heavy use of compressed-domain manipulation and adaptive sampling schemes to accomplish various sensitivity analysis tasks and queries over TT-based surrogate models. Our methods are compact and scalable to many dimensions. To the best of our knowledge, ours is the first approach that can handle and use all sensitivity indices at once.

The proposed framework is highly flexible in several aspects. We have shown tensor decompositions to be suitable for both cardinal and ordinal/categorical data, for either dense or sparse samples, and for any dimensionality. New dimensions can be stacked or concatenated to existing decompositions. Tensors are in all ways just as versatile for 3 and more variables as matrices are for bivariate data. The field is growing both theoretically and in applications, with several recent tensor developments pointing to pressing trends like deep learning or the Internet of things.

The most important feature we demonstrated is the ability to modify, query

and analyze the data in its compressed form. Multilinearity is straightforward to apply and inherent to all exploited formats: it works along the factor columns for CP/Tucker, and slice-wise along the cores for TT. This already allows for differentiation, rectangle integration, and any kind of linear transform. The second generation of tools arises from adaptive sampling and provides advanced operations that are important for analysis and visualization, such as element-wise functions and global optimization. In this context, the TT format deserves a special mention. It is the only model considered in this thesis that a) grows well with dimensionality; b) has stable decomposition algorithms; and c) supports a wide variety of scalable sampling and manipulation techniques. It is thus a safe model choice applicable in most situations. Tucker is more competitive for $N = 3$; however, both TT and Tucker are particular cases of TT-Tucker (Ch. 6), which we believe gives a positive answer to RQ2 as a good "universal model" for the goals we pursued.

Several mathematical tools pervade the presented analysis and visualization pipeline, and several ideas that arose when facing a specific challenge were often applicable to other settings as well. These include:

- The concept of tensor rank is key to all algorithms developed. More ranks allow matching the available data better, but entail higher computational compression/decompression costs, may require higher number of samples, and often result in overfitting. Fewer ranks produce a coarser approximation of the ground-truth, but reduce the computational burden and may generalize better –in the sense of more meaningful visual features, smoother interpolation, etc.

- Several compressed-domain operations that are beneficial in dense data are similarly useful for analyzing surrogate models fitted on sparse data sets. For example, the inclusion-exclusion principle translates naturally to any tensor format. We used it to both query tensor-compressed SATs and IHs (Ch. 5) and to efficiently compute arbitrary sensitivity indices in a compact form (Ch. 7).

- Global optimization is the ideal tool for custom-defined queries on both plain tensor models (Ch. 6) and derived tensors (namely the Sobol tensor and variants, Ch. 7). See also the next section for more future possibilities.

8.2 Future Work

8.2.1 High-dimensional Compression

Several extensions and opportunities related to high dimensional dense data are still open. For example, a multiresolution Tucker octree could be defined for

time-varying volumes, although it is not clear a priori how to achieve frame reconstruction real-time in such an architecture. In addition, a compressor using adaptive quantization (as in Ch. 3) in the TT or TT-Tucker formats, instead of Tucker, could have a great potential for high-dimensional data reduction. However, such formats are more asymmetric than CP or Tucker: dimension ordering matters as argued in Ch. 5, and variables that are given a central core tend to need more ranks. In addition, TT cannot be thresholded directly like the Tucker core, and tools like *core orthogonalization* [Oseledets, 2011] will play a necessary role. These difficulties will require careful investigation.

8.2.2 Categorical Visualization

The tensors we built had always discrete axis, and the last tensor we sampled in Ch. 6, the GEMM experiment, had categorical variables. However, several further lines of research for categorical data are approachable on the visualization side. We believe many diagrams that are suitable for categorical data (mosaic plots, parallel sets, bar charts, etc.) will combine excellently with tensor surrogates, especially for high numbers of dimensions. Such diagrams can be potentially enhanced by various analysis tasks that are feasible in the tensor framework, e.g. via global optimization, as similarly to what we did. After all, tensor decompositions were born precisely for categorical data (in the field of psychometrics).

8.2.3 Multivalued Data

We wish to investigate more deeply TT surrogates for multi-valued models. Rather than training a separate model per individual output, we would like to work on a single tensor with an extra dimension (core) to index the outputs. One may then perform joint TT visualization and analysis, e.g. reconstructing linked plots at once in an ensemble fashion, or obtaining a Sobol TT with an extra indexing dimension that allows conjunct sensitivity queries over the multiple outputs.

8.2.4 Smooth Tensor Completion

We have used tensor completion, which is a powerful paradigm for categorical data in non-adaptive situations. However, its application to continuous variables is more delicate. Techniques such as DCT or PCE, for example, incorporate smoothness-based priors by fixing their bases. In the future we will increasingly use smoothness-aware completion strategies, for example using similarity matrices [Narita et al., 2011]. Such techniques will allow us to build low-rank tensors from non-adaptive, smooth data sets without having to convert from another model, e.g. PCE.

8.2.5 Content-based Surrogate Navigation

The global optimization algorithm exploited in Ch. 6 and Ch. 7 opens up many avenues of research. One such direction is query-by-sketch for surrogate models, whereby a user would search and identify trends in the model by drawing curves or shapes. For example, he/she could ask which parameter combinations result in ascending/descending patterns in the surrogate's output. Such an application would be of great aid in knowledge extraction and interactive exploration, as it would allow the user to quickly and intuitively jump to interesting focus points across the whole domain of variables.

APPENDIX A

DECOMPOSITION ALGORITHMS

Three different kinds of tensor decomposition algorithms exist depending on the nature of the input: dense data, sparse data, and adaptive sampling (i.e. when we can sample on demand, but not the whole tensor).

A.1 From Dense Data

A.1.1 CP

The first question is how to select the number of CP ranks R; CORCONDIA [Bro and Kiers, 2003] is an algorithm that performs a consistency diagnostic to compare different numbers of components. If the desired number of ranks is known or estimated, there is an alternating least squares (ALS) algorithm known as higher-order power method (HOPM) [Kolda and Bader, 2009]. The algorithm fixes N-1 matrices and then solves for the remaining one; this is cyclically done for every matrix. While a rank-R decomposition is not always unique, in practice ALS finds very good solutions for most purposes concerning visual data sets. HOPM is detailed in Alg. 10.

A.1.2 Tucker

The first proposed decomposition algorithm for the Tucker model was the so-called *Tucker1* [Tucker, 1966]. It is a truncation of a decomposition nowadays known as HOSVD, which is in many senses a higher-order generalization of the

Algorithm 10 The higher-order power method (HOPM) computes a rank-R CP decomposition of a tensor $\mathcal{T}$ in K iterations.

$\boldsymbol{\lambda} := \text{zeros}(R)$
$\mathbf{U}^{(n)} := \text{init}() \ \forall n = 1, \ldots, N$
for $k = 1, \ldots, K$ **do**
  for $n = 1, \ldots, N$ **do**
    $\mathbf{V} := \mathbf{U}^{(1)^T}\mathbf{U}^{(1)} \circ \cdots \circ \mathbf{U}^{(N)^T}\mathbf{U}^{(N)}$ {The term $\mathbf{U}^{(n)^T}\mathbf{U}^{(n)}$ is skipped}
    $\mathbf{U}^{(n)} := \mathbf{T}_{(n)}(\mathbf{U}^{(N)} \odot \cdots \odot \mathbf{U}^{(1)})\mathbf{V}^{\dagger}$ {The term $\mathbf{U}^{(n)}$ is skipped}
    for $r = 0, \ldots, R-1$ **do**
      $\boldsymbol{\lambda}[r] := \|\mathbf{U}^{(n)}[:, r]\|$
      $\mathbf{U}^{(n)}[:, r] = \mathbf{U}^{(n)}[:, r]/\boldsymbol{\lambda}[r]$
    end for
  end for
end for
return $[[\lambda; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}]]$

2D matrix SVD [de Lathauwer et al., 2000a]. However, such a truncation is not theoretically optimal in terms of best fit (although it is proven to be close [de Lathauwer et al., 2000b]). For this reason, ALS algorithms were designed in order to improve upon it: [Kroonenberg and Leeuw, 1980] contributed *TUCKALS3*, the first ALS procedure for finding a least-squares 3D Tucker tensor approximation. Later ALS approaches such as [Kroonenberg, 1983], [Ten Berge et al., 1987] or the higher-order orthogonal iteration (HOOI) [de Lathauwer et al., 2000b] were developed to optimize or accelerate the basic TUCKALS. We use the version of HOOI as provided in [Kolda and Bader, 2009] (Alg. 11). It can be seen as an extension of the HOSVD initialization, since each matrix is populated with singular values at each mode optimization.

Algorithm 11 The higher-order orthogonal iteration (HOOI) computes a rank-$(R_1, \ldots, R_N)$ Tucker decomposition of a tensor $\mathcal{T}$ in K iterations.

$\mathbf{U}^{(n)} := \text{init}() \ \forall n = 1, \ldots, N$
for $k = 1, \ldots, K$ **do**
  for $n = 1, \ldots, N$ **do**
    $\mathcal{P} := \mathcal{T} \times_1 \mathbf{U}^{(1)^T} \times_2 \cdots \times_N \mathbf{U}^{(N)^T}$ {The term $\mathbf{U}^{(n)^T}$ is skipped}
    $\mathbf{U}^{(n)} := R_n$ leading eigenvectors of $\mathbf{P}_{(n)}{\mathbf{P}_{(n)}}^T$
  end for
end for
$\mathcal{B} := \mathcal{T} \times_1 \mathbf{U}^{(1)^T} \times_2 \cdots \times_N \mathbf{U}^{(N)^T}$
return $[[(\mathcal{B}; \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)})]]$

A.1.3 TT

A full tensor can be TT-decomposed via the *TT-SVD* algorithm [Oseledets, 2011], which successively unfolds the data, computes the SVD on the result, and truncates it. This algorithm is non-iterative and separates one dimension at a time. It has cost $O(NIR^3)$; see Alg. 12 for a version with fixed ranks. Alternatively, the ranks can be determined on the fly in terms of a user defined relative error $0 \leq \epsilon \leq 1$. The algorithm guarantees that the final error will not be larger than ϵ, and in fact it is usually smaller. For a detailed description of the rank-adaptive TT-SVD and its theoretical error bounds we refer the reader to the original paper [Oseledets, 2011].

Algorithm 12 TT-SVD algorithm [Oseledets, 2011] to build a rank-$(R_1, \ldots, R_{N-1})$ TT decomposition.

$\mathbf{M} := \text{unfold}(\mathcal{T}, I_1 \times I_2 \cdots I_N)$
for $n = 1, \ldots, N-1$ **do**
  $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} := \text{SVD}(\mathbf{M})$
  $\mathcal{T}^{(n)} := \text{reshape}(\mathbf{U}[:, 0{:}R_n - 1], R_{n-1} \times I_n \times R_n)$ $\{R_0 = 1$ by convention$\}$
  $\mathbf{M} := \text{reshape}(\mathbf{\Sigma}[0{:}R_n - 1, 0{:}R_n - 1] \cdot \mathbf{V}[:, 0{:}R_n - 1]^T, R_n I_{n+1} \times R_{n+1})$
end for
$\mathcal{T}^{(N)} := \text{reshape}(\mathbf{M}, R_{N-1} \times I_N \times 1)$
return $[[\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N)}]]$

A.2 From a Fixed Sparse Sample Set

The problem of tensor completion arises often in signal processing and machine learning, and is a useful tool to handle e.g. corrupted regions in images or volumes, or learn unknown entries in a parameter space from a limited sparse set of observations. In the same way that algorithms for dense tensor decomposition generalize earlier matrix techniques, also the tensor completion framework is based on previous 2D results. The field is related to multidimensional compressive sensing [Caiafa and Cichocki, 2013].

More formally, suppose $\Omega \subset \{1, \ldots, I_1\} \times \cdots \times \{1, \ldots, I_N\}$ is a set encoding certain tensor entries and $f|_\Omega$ represents the known values at these entries. Then, a recovered tensor $\mathcal{T}$ is sought such that a) $\mathcal{T}(\Omega) = f|_\Omega$; and b) $\mathcal{T}$ has low rank. Condition a) is often relaxed and incorporated as a penalization term into a score function (e.g. Lagrangian relaxation) in the presence of noise, or whenever accuracy loss is acceptable to certain extent in exchange for a sufficiently low-parametric representation. Condition b) also accepts different formulations: one may impose that the ranks (under the tensor rank definition of choice) are

upper-bounded or even fixed to some value. For instance, the Riemannian optimization framework fixes the number of ranks and considers the set of tensors in $\mathbb{R}^{I_1 \times \cdots \times I_N}$ that have such number of ranks [Steinlechner, 2015]. Then, a score function is minimized over such sets. Riemannian methods require that such set forms a manifold. This is the case for the Tucker, TT or HT rank definition, but not for CP: the set of rank-R CP tensors is not closed, since there exist sequences of rank-R tensors such that their limit has rank other than R (the so-called border rank).

An alternative strategy is to relax the rank constraint, for example by using the matrix nuclear norm $\|\mathbf{A}\|_*$ as the minimization target. It is defined as the sum of its singular values (in absolute value) and is known to be the tightest convex envelope of the rank function for the 2D case. A number of algorithms generalize the norm $\|\cdot\|_*$ to the tensor case and then make use of convex solvers to find the recovered tensor $\mathcal{T}$. A comprehensive survey on tensor completion (and more general techniques to produce low-rank decompositions) was compiled by [Grasedyck et al., 2013].

In this thesis we have implemented and used an ALS completion algorithm for the TT format. This strategy seeks a solution in the manifold of fixed TT ranks; it is based on the ALS detailed in [Steinlechner, 2015]. Optimizing each core requires first computing two sequences of matrix-matrix products (the left and right *product chains*) for each training sample. Our implementation uses memoization from core to core to avoid recomputing these chains every time; see Alg. 13.

A.3 Adaptive Sampling

The quest for adaptive tensor sampling schemes started, unsurprisingly, in the matrix case. The *CUR* matrix decomposition (also known as *pseudo-skeleton* or, alternatively, *column-row factorization*) reconstructs a matrix from a well-structured subset of its samples, namely its rows and columns [Bebendorf, 2000], [Tyrtyshnikov, 2000]. CUR takes its name from the three matrices that take part in the approximate reconstruction of an input $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2}$:

$$\mathbf{A} \approx \mathbf{C}\mathbf{U}^{-1}\mathbf{R} \tag{A.1}$$

where $\mathbf{C} \in \mathbb{R}^{I_1 \times R}$, $\mathbf{U} \in \mathbb{R}^{R \times R}$, and $\mathbf{R} \in \mathbb{R}^{R \times I_2}$. The matrices $\mathbf{C}$ and $\mathbf{R}$ consist of R selected columns and rows from $\mathbf{A}$, respectively, while $\mathbf{U}$ is the intersection of $\mathbf{C}$ and $\mathbf{R}$. This is a crucial difference with SVD, which defines its own transform vectors. Thanks to the CUR decomposition, only $O(IR)$ elements (instead of $O(I^2)$) must be sampled and stored to learn and represent the full matrix $\mathbf{A}$. If $\mathbf{A}$ has rank R, then $\mathbf{C}$ and $\mathbf{R}$ can be chosen so that Eq. A.1 is exact. Otherwise, the best possible R-approximation comes from selecting the rows and columns that

Algorithm 13 TT-ALS rank-R completion with K sweeps from a set $(\mathbf{X}, \mathbf{y})$ of known samples. $\mathbf{X}$ is a matrix of size $P \times N$ and $\mathbf{y}$ is a vector of size P. The initial solution is $[[\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(N)}]]$. This dynamic algorithm maintains two arrays $\mathcal{L}$ and $\mathcal{R}$ to store the left and right product chains for all sample points in $\mathbf{X}$. For each data point $(\mathbf{x}, y)$ and n-th core these chains equal $\mathbf{l} = \prod_{i<n} \mathcal{T}^{(i)}[x_i]$ and $\mathbf{r} = \prod_{i>n} \mathcal{T}^{(i)}[x_i]$. Then, we must find the core slice $\mathcal{T}^{(n)}$ that minimizes $\|\mathbf{l} \cdot \mathcal{T}^{(n)}[x_n] \cdot \mathbf{r} - y\| = \|(\mathbf{r}^T \otimes \mathbf{l}) \cdot \mathrm{vec}(\mathcal{T}^{(n)}[x_n]) - y\|$ (linear least-squares problem).

{Initialization}
for $n = 1, \ldots, N-1$ **do**
  $\mathcal{L}^{(n)} := \mathrm{ones}(1, P, 1)$
  $\mathcal{R}^{(n+1)} := \mathrm{ones}(1, P, 1)$
end for
for $n = N-2, \ldots, 1$ **do**
  for $p = 1, \ldots, P$ **do**
    $\mathcal{R}^{(n)}[:, p, :] := \mathcal{T}^{(n+1)}[:, \mathbf{X}[p, n+1], :] \cdot \mathcal{R}^{(n+1)}[:, p, :]$
  end for
end for
{K sweeps}
for $k = 1, \ldots, K$ **do**
  {Left-to-right sweep}
  for $n = 1, \ldots, N-1$ **do**
    {Optimize the n-th core, slice-by-slice}
    for $i = 1, \ldots, I_n$ **do**
      $\mathbf{A} :=$ matrix of size $0 \times R^2$
      for $p \mid \mathbf{X}[p, n] = i$ **do**
        $\mathrm{addRow}(\mathbf{A}, \mathcal{R}^{(n)}[:, p, :]^T \otimes \mathcal{L}^{(n)}[:, p, :])$
      end for
      $\mathbf{b} := \{\mathbf{y}[p] \mid \mathbf{X}[p, n] = i\}$
      $\mathbf{S} := \mathrm{leastSquares}(\mathbf{A}, \mathbf{b})$ {Best core slice for all points it affects}
      $\mathcal{T}^{(n)}[:, i, :] := \mathrm{reshape}(\mathbf{S}, R \times R)$
    end for
    for $p = 1, \ldots, P$ **do**
      {Update next $\mathcal{L}$ by multiplying with the newly computed core $\mathcal{T}^{(n)}$}
      $\mathcal{L}^{(n+1)}[:, p, :] := \mathcal{L}^{(n)}[:, p, :] \cdot \mathcal{T}^{(n)}[:, \mathbf{X}[p, n], :]$
    end for
  end for
  {The right-to-left sweep works analogously, with $n = N, \ldots, 2$ and updating $\mathcal{R}$ instead of $\mathcal{L}$}
end for

produce an intersection $\mathbf{U}$ with the largest determinant in modulus [Goreinov and Tyrtyshnikov, 2001]. Geometrically, given a set P of points in an N-dimensional space (with $|P| > N$) we wish to select a set $Q \subset P$ (with $|Q| = N$) such that the parallelepiped spanned by the vectors in Q has the largest possible volume. The *maxvol* algorithm [Tyrtyshnikov, 2000], [Goreinov et al., 2008] is a very effective adaptive heuristic that proceeds by adding a few rows and columns at a time. Maxvol and its variant the *rectangular maxvol* have other uses such as feature selection or finding maximum elements in matrices [Mikhalev and Oseledets, 2015].

CUR and maxvol have been generalized to N dimensions in the form of the so-called *adaptive cross approximation* (ACA). Some early contributions in this direction include [Espig et al., 2009] for CP and [Oseledets et al., 2008] and [Caiafa and Cichocki, 2010] for the Tucker format, and have been used for e.g. volume completion in human brain data sets. ACA has been later blended in with the TT formulation [Oseledets and Tyrtyshnikov, 2010b]. It has since then emerged as a powerful paradigm for efficient multidimensional data sampling and manipulation. Only $O(NIR^2)$ samples must be taken in order to recover (interpolate) a full I^N TT tensor, i.e. a number proportional to the number of coefficients in the compressed tensor [Savostyanov and Oseledets, 2011]. Analogously to the 2D case, many useful transformations in a TT (for example, computing element-wise functions) require manipulating only the cross entries and fibers of the input. In this thesis we use the implementation from the ttpy toolbox [ttp,] (open-source, written in Python), since it also includes many manipulation routines for the TT format.

Tab. A.1 lists many important tensor completion and adaptive sampling algorithms that have been proposed for various formats and settings.

Method	Tensor Type	Adaptive?
[Bebendorf, 2000]	Matrix	Yes
[Tyrtyshnikov, 2000]	Matrix	Yes
[Candès and Recht, 2009]	Matrix	No
[Espig et al., 2009]	CP	Yes
[Espig et al., 2011]	CP	No
[Oseledets et al., 2008]	Tucker	Yes
[Goreinov, 2008]	Tucker	Yes
[Caiafa and Cichocki, 2010]	Tucker	Yes
[Kressner et al., 2013]	Tucker	No
[Ballani et al., 2013]	HT	Yes
[Ballani and Grasedyck, 2015]	HT	Yes
[Silva and Herrmann, 2015]	HT	No
[Oseledets and Tyrtyshnikov, 2010b]	TT	Yes
[Savostyanov and Oseledets, 2011]	TT	Yes
[Steinlechner, 2015]	TT	No
[Grasedyck et al., 2015]	TT	No
[Bengua et al., 2016]	TT	No

Table A.1: *Several completion and adaptive sampling algorithms, from the matrix case up to more recent models.*

APPENDIX B

TENSOR DECOMPOSITION SOFTWARE

Multiway array data structures are often recognized readily by modern languages and libraries; for instance, both Python (via NumPy) and MATLAB support arrays of arbitrary dimensionality. The C++ header-only library Eigen [eig,] supports them via the Eigen::Tensor core module; TensorFlow [ten, b] is another option. A wide range of programming languages provide operations at the tensor level including slicing, reshaping, tensor-scalar operations, tensor contraction, etc. Next we list only the subset of libraries that can actually compute decompositions out of tensor arrays (we have compiled a periodically updated list in [pub,]).

Kolda and Bader's Tensor Toolbox [Bader et al., 2015] is one of the earliest packages for MATLAB and features sparse and dense CP and Tucker decompositions. Tensorlab [Sorber et al.,] supports these and TT, as well as other highly structured and hybrid representations (optionally, with constraints such as non-negativity). It also includes several completion algorithms. The Laboratory for Tensor Decomposition and Analysis (TDALAB) [tda,] and TensorBox [ten, a] are two MATLAB libraries tailored (among others) to blind source separation (BSS). MATLAB Tensor Tools (MTT) [mtt,] is oriented towards computer vision applications and supports non-negative tensor factorization (NTF) as well as incremental decomposition procedures, including CP and Tucker defined on sliding windows for exploiting local temporal correlation such as video scenes in foreground/background segmentation settings. Regarding tensor completion tasks,

GeomCG [geo,] provides routines for Tucker format inpainting within the Riemannian optimization framework on the manifold of tensors with fixed multilinear rank. The Riemannian framework for the case of tensors in TT form is supported by the TTeMPS Toolbox [tte,], while the Hierarchical Tucker Toolbox [htt,] is concerned with the HT format.

In Python, the library scikit-tensor [sci,] obtains CP and Tucker among others for both sparse and dense tensors. NTFLib [ntf,], on the other hand, emphasizes non-negative factorizations focusing on sparse data. TensorLy [ten, c] provides both non-negative and general CP and Tucker decompositions for dense data and dives further into learning and vision applications by contributing tensorized ridge regression algorithms for these formats. For the TT model, ttpy [ttp,] provides basic decomposition routines as well as rounding, adaptive sampling of black-box tensors, eigenvalue and linear system solvers, etc. Many of its core features are actually implemented in FORTRAN (and there is also an alternative MATLAB implementation of the toolbox [ttt,]). The Python TensorToolbox [pyt,] provides functional (continuous) versions of the TT format. Polara [pol,] implements sparse tensor factorizations with a focus on recommender systems.

Finally, SPLATT [spl,] can be used to compute CP out of sparse tensors in C++. The vmmlib [vmm, c] supports CP and Tucker decompositions of dense tensors, and uses BLAS and LAPACK parallel routines to optimize matrix operations. The deep learning framework Caffe2 [caf,] supports TT-compressed fully connected layers. There is also a C++ library for the TT format [cpl,] that exploits parallelism via the OpenMP API.

BIBLIOGRAPHY

[hur,] 2004 SciVis Contest.
http://sciviscontest-staging.ieeevis.org/2004/data.html.

[cpl,] C++ TT library.
https://bitbucket.org/dzheltkov/c-tt-library.

[caf,] Caffe: A deep learning framework.
http://caffe.berkeleyvision.org/.

[ces,] Community Earth System Model by the National Center for Atmospheric Research.
http://www.cesm.ucar.edu/index.html.

[cup,] CuPy: A numpy-compatible matrix library accelerated by CUDA.
https://cupy.chainer.org/.

[eig,] Eigen: a C++ template, header-only library for linear algebra.
http://eigen.tuxfamily.org/.

[wat,] Furong Waterfalls. https://commons.wikimedia.org/wiki/File:1furongpanorama2012.jpg.

[geo,] GeomCG: Tensor completion by Riemannian optimization.
http://anchp.epfl.ch/geomCG.

[htt,] HT Toolbox: A MATLAB toolbox for the construction and manipulation of tensors in the hierarchical Tucker format.
http://anchp.epfl.ch/htucker.

[iap,] IAPR-TC18 data sets.
http://www.tc18.org/code_data_set/3D_images.php.

[jht,] Johns Hopkins Turbulence Database.
http://turbulence.pha.jhu.edu/newcutout.aspx.

[pub,] A list of public tensor decomposition software.
http://github.com/rballester/tensor_notes/blob/master/implementations.md.

[mtt,] Matlab Tensor Tools: Matrix and tensor tools for computer vision.
http://github.com/andrewssobral/mtt.

[ntf,] NTFLib: Sparse beta-divergence tensor factorization library.
http://github.com/mnick/scikit-tensor.

[pol,] Polara: a recommender system and evaluation framework.
http://github.com/Evfro/polara.

[vol,] Real world medical datasets.
http://volvis.org/.

[vmm, a] Research datasets from the Visualization and Multimedia Lab.
http://www.ifi.uzh.ch/vmml/research/datasets.html.

[sci,] scikit-tensor: a Python module for multilinear algebra and tensor factorizations.
http://github.com/mnick/scikit-tensor.

[spl,] SPLATT: The surprisingly parallel sparse tensor toolkit.
http://shaden.io/splatt.html.

[tda,] TDALAB: Laboratory for tensor decomposition and analysis.
http://www.bsp.brain.riken.jp/TDALAB/.

[ten, a] TensorBox: a MATLAB toolbox for tensor decompositions.
http://www.bsp.brain.riken.jp/~phan/index.html#tensorbox.

[ten, b] Tensorflow – an open source software library for machine intelligence.
http://www.tensorflow.org/.

[ten, c] TensorLy: a fast and simple Python library for tensor learning. https://github.com/tensorly/tensorly.

[pyt,] TensorToolbox: tensor train and spectral tensor train decomposition. https://pypi.python.org/pypi/TensorToolbox/.

[ttt,] TT-Toolbox: the MATLAB tensor train toolbox. http://github.com/oseledets/TT-Toolbox.

[tte,] TTeMPS: A toolbox for tensor train / matrix product states. http://anchp.epfl.ch/TTeMPS.

[ttp,] ttpy: Python implementation of the TT-toolbox. http://github.com/oseledets/ttpy.

[vmm, b] Visualization and MultiMedia Lab's Research Data Sets. http://www.ifi.uzh.ch/en/vmml/research/datasets.html.

[vmm, c] vmmlib: A Vector and Matrix Math Library. http://vmml.github.io/vmmlib/.

[Abualrub et al., 2017] Abualrub, T., Jarrah, A., Kallel, S., and Sulieman, H. (2017). *Mathematics Across Contemporary Sciences: AUS-ICMS, Sharjah, UAE, April 2015*. Springer Proceedings in Mathematics & Statistics. Springer International Publishing.

[Amirkhanov et al., 2010] Amirkhanov, A., Heinzl, C., Reiter, M., and Gröller, M. E. (2010). Visual optimality and stability analysis of 3DCT scan positions. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1477–1486.

[Anand et al., 2012] Anand, A., Dang, T. N., and Wilkinson, L. (2012). Visual pattern discovery using random projections. In *Proceedings IEEE Conference on Visual Analytics Science and Technology*, pages 43–52.

[Bader et al., 2015] Bader, B. W., Kolda, T. G., et al. (2015). MATLAB Tensor Toolbox version 2.6. http://www.sandia.gov/~tgkolda/TensorToolbox/.

[Ballani and Grasedyck, 2015] Ballani, J. and Grasedyck, L. (2015). Hierarchical tensor approximation of output quantities of parameter-dependent PDEs. *SIAM Journal on Uncertainty Quantification*, 3(1):852–872.

[Ballani et al., 2013] Ballani, J., Grasedyck, L., and Kluge, M. (2013). Black box approximation of tensors in hierarchical tucker format. *Linear Algebra and its Applications*, 438(2):639 – 657.

[Ballester-Ripoll et al., 2017] Ballester-Ripoll, R., Lindstrom, P., and Pajarola, R. (2017). TTHRESH: Tensor compression for multidimensional visual data. *In preparation*.

[Ballester-Ripoll and Pajarola, 2015] Ballester-Ripoll, R. and Pajarola, R. (2015). Lossy volume compression using Tucker truncation and thresholding. *The Visual Computer*, pages 1–14.

[Ballester-Ripoll and Pajarola, 2016] Ballester-Ripoll, R. and Pajarola, R. (2016). Compressing bidirectional texture functions via tensor train decomposition. In *Proceedings Pacific Graphics Short Papers*.

[Ballester-Ripoll and Pajarola, 2017] Ballester-Ripoll, R. and Pajarola, R. (2017). Tensor decompositions for integral histogram compression and look-up. *IEEE Transactions on Visualization and Computer Graphics*, PP:1–12.

[Ballester-Ripoll et al., 2016] Ballester-Ripoll, R., Paredes, E. G., and Pajarola, R. (2016). A surrogate visualization model using the tensor train format. In *SIGGRAPH ASIA 2016 Symposium on Visualization*, pages 13:1–13:8.

[Ballester-Ripoll et al., 2017] Ballester-Ripoll, R., Paredes, E. G., and Pajarola, R. (2017). Sobol tensor trains for global sensitivity analysis. *ArXiv e-prints*.

[Ballester-Ripoll et al., 2017] Ballester-Ripoll, R., Steiner, D., and Pajarola, R. (2017). Multiresolution volume filtering in the tensor compressed domain. *IEEE Transactions on Visualization and Computer Graphics*, PP:1–14.

[Ballester-Ripoll et al., 2015] Ballester-Ripoll, R., Suter, S. K., and Pajarola, R. (2015). Analysis of tensor approximation for compression-domain volume visualization. *Computers & Graphics*, 47:34–47.

[Balsa Rodríguez et al., 2013] Balsa Rodríguez, M., Gobbetti, E., Guitián, J. A. I., Makhinya, M., Marton, F., Pajarola, R., and Suter, S. K. (2013). A survey of compressed GPU direct volume rendering. In *Eurographics State of The Art Reports (STAR)*.

[Balsa Rodríguez et al., 2014] Balsa Rodríguez, M., Gobbetti, E., Iglesias Guitián, J. A., Makhinya, M., Marton, F., Pajarola, R., and Suter, S. K. (2014). State-of-the-art in compressed GPU-based direct volume rendering. *Computer Graphics Forum*, 33(6):77–100.

[Bebendorf, 2000] Bebendorf, M. (2000). Approximation of boundary element matrices. *Numerische Mathematik*, 86(4):565–589.

[Bengua et al., 2016] Bengua, J. A., Phien, H. N., Tuan, H. D., and Do, M. N. (2016). Efficient tensor completion for color image and video recovery: Low-rank tensor train. *CoRR*, abs/1606.01500.

[Berger et al., 2012] Berger, R., Dubuisson, S., and Gonzales, C. (2012). Fast multiple histogram computation using Kruskal's algorithm. In *Proceedings IEEE International Conference on Image Processing*, pages 2373–2376.

[Bergner et al., 2013] Bergner, S., Sedlmair, M., Moller, T., Abdolyousefi, S. N., and Saad, A. (2013). Paraglide: Interactive parameter space partitioning for computer simulations. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1499–1512.

[Bigoni, 2015] Bigoni, D. (2015). *Uncertainty Quantification with Applications to Engineering Problems*. PhD thesis.

[Bigoni et al., 2016] Bigoni, D., Engsig-Karup, A., and Marzouk, Y. (2016). Spectral tensor-train decomposition. *SIAM Journal on Scientific Computing*, 38(4):A2405–A2439.

[Brecheisen et al., 2009] Brecheisen, R., Vilanova, A., Platel, B., and ter Haar Romeny, B. (2009). Parameter sensitivity visualization for dti fiber tracking. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1441–1448.

[Bro and Kiers, 2003] Bro, R. and Kiers, H. A. (2003). A new efficient method for determining the number of components in PARAFAC models. *Journal of Chemometrics*, 16:387–400.

[Cabot and Cook,] Cabot, W. H. and Cook, A. W. Reynolds number effects on Rayleigh-Taylor instability with possible implications for type Ia supernovae. *Nature Physics*, 2:562 – 568.

[Caiafa and Cichocki, 2010] Caiafa, C. and Cichocki, A. (2010). Generalizing the column–row matrix decomposition to multi-way arrays. *Linear Algebra Applications*, 433(3):557–573.

[Caiafa and Cichocki, 2013] Caiafa, C. F. and Cichocki, A. (2013). Multidimensional compressed sensing and their applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 3(6):355–380.

[Candès and Recht, 2009] Candès, E. J. and Recht, B. (2009). Exact matrix completion via convex optimization. *Foundations on Computational Mathematics*, 9(6):717–772.

[Chan, 2006] Chan, W. W.-Y. (2006). A survey on multivariate data visualization. Technical report, Department of Computer Science and Engineering. Hong Kong University of Science and Technology.

[Chen et al., 2016] Chen, M., Feixas, M., Viola, I., Bardera, A., Shen, H., and Sbert, M. (2016). *Information Theory Tools for Visualization*. AK Peters Visualization Series. CRC Press.

[Chen et al., 2014] Chen, Y.-L., Hsu, C.-T., and Liao, H.-Y. M. (2014). Simultaneous tensor decomposition and completion using factor priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):577–591.

[Cichocki et al., 2016] Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. P. (2016). Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends in Machine Learning*, 9(4-5):249–429.

[Clyne et al., 2007] Clyne, J., Mininni, P., Norton, A., and Rast, M. (2007). Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. *New Journal of Physics*, 9(8):301.

[Costantini et al., 2008] Costantini, R., Sbaiz, L., and Süsstrunk, S. (2008). Higher order SVD analysis for dynamic texture synthesis. *IEEE Transactions on Image Processing*, pages 42–52.

[Crow, 1984] Crow, F. C. (1984). Summed-area tables for texture mapping. In *Proceedings ACM SIGGRAPH*, pages 207–212.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 886–893.

[De Lathauwer, 2008a] De Lathauwer, L. (2008a). Decompositions of a higher-order tensor in block terms — part i: Lemmas for partitioned matrices. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1022–1032.

[De Lathauwer, 2008b] De Lathauwer, L. (2008b). Decompositions of a higher-order tensor in block terms – Part II: Definitions and uniqueness. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1033–1066.

[de Lathauwer, 2009] de Lathauwer, L. (2009). A survey of tensor methods. In *Proceedings IEEE International Symposium on Circuits and Systems*, pages 2773–2776.

[de Lathauwer et al., 2000a] de Lathauwer, L., de Moor, B., and Vandewalle, J. (2000a). A multilinear singular value decomposition. *SIAM Journal of Matrix Analysis and Applications*, 21(4):1253–1278.

[de Lathauwer et al., 2000b] de Lathauwer, L., de Moor, B., and Vandewalle, J. (2000b). On the best rank-1 and rank-($R_1, R_2, ..., R_N$) approximation of higher-order tensors. *SIAM Journal of Matrix Analysis and Applications*, 21(4):1324–1342.

[de Silva and Lim, 2008] de Silva, V. and Lim, L.-H. (2008). Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1084–1127.

[Di and Cappello, 2016] Di, S. and Cappello, F. (2016). Fast error-bounded lossy hpc data compression with SZ. In *International Parallel and Distributed Processing Symposium*, pages 730–739.

[Dolgov et al., 2014] Dolgov, S. V., Khoromskij, B. N., Litvinenko, A., and Matthies, H. G. (2014). Computation of the response surface in the tensor train data format. arXiv preprint 1406.2816.

[Dolgov and Savostyanov, 2014] Dolgov, S. V. and Savostyanov, D. V. (2014). Alternating minimal energy methods for linear systems in higher dimensions. *SIAM Journal on Scientific Computing*, 36(5):A2248–A2271.

[dos Santos Amorim et al., 2012] dos Santos Amorim, E. P., Brazil, E. V., II, J. D., Joia, P., Nonato, L. G., and Sousa, M. C. (2012). iLAMP: Exploring high-dimensional spacing through backward multidimensional projection. In *Proceedings IEEE Conference on Visual Analytics Science and Technology*, pages 53–62.

[Efron and Stein, 1981] Efron, B. and Stein, C. (1981). The jackknife estimate of variance. *Ann. Statist.*, 9(3):586–596.

[Eilemann et al., 2009] Eilemann, S., Makhinya, M., and Pajarola, R. (2009). Equalizer: A scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):436–452.

[Espig et al., 2009] Espig, M., Grasedyck, L., and Hackbusch, W. (2009). Black box low tensor-rank approximation using fiber-crosses. *Constructive Approximation*, 30(3):557–597.

[Espig et al., 2011] Espig, M., Hackbusch, W., Litvinenko, A., Matthies, H. G., and Zander, E. (2011). Efficient analysis of high dimensional data in tensor formats. *Lecture notes in computational science and engineering*.

[Falch and Elster, 2017] Falch, T. L. and Elster, A. C. (2017). Machine learning-based auto-tuning for enhanced performance portability of opencl applications. *Concurrency and Computation: Practice and Experience*, 29(8).

[Filipović and Jukić, 2015] Filipović, M. and Jukić, A. (2015). Tucker factorization with missing data with application to low-n-rank tensor completion. *Multidimensional Syst. Signal Process.*, 26(3):677–692.

[Fock, 2014] Fock, E. (2014). Global sensitivity analysis approach for input selection and system identification purposes – a new framework for feedforward neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1484–1495.

[Fout and Ma, 2007] Fout, N. and Ma, K.-L. (2007). Transform coding for hardware-accelerated volume rendering. *IEEE Transaction on Visualization and Computer Graphics*, 13(6):1600–1607.

[Gerber et al., 2010] Gerber, S., Bremer, P.-T., Pascucci, V., and Whitaker, R. (2010). Visual exploration of high dimensional scalar functions. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1271–1280.

[Gobbetti et al., 2012] Gobbetti, E., Iglesias Guitián, J., and Marton, F. (2012). COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks. *Computer Graphics Forum*, 31(3):1315–1324.

[Goreinov, 2008] Goreinov, S. A. (2008). On cross approximation of multi-index arrays. *Doklady Mathematics*, 77(3):404–406.

[Goreinov et al., 2008] Goreinov, S. A., Oseledets, I. V., Savostyanov, D. V., Tyrtyshnikov, E. E., and Zamarashkin, N. L. (2008). How to find a good submatrix. Research Report 08-10, ICM HKBU, Kowloon Tong, Hong Kong.

[Goreinov and Tyrtyshnikov, 2001] Goreinov, S. A. and Tyrtyshnikov, E. E. (2001). The maximal-volume concept in approximation by low-rank matrices. *Contemporary Mathematics*, 280:47–51.

[Grasedyck et al., 2015] Grasedyck, L., Kluge, M., and Krämer, S. (2015). Variants of alternating least squares tensor completion in the tensor train format. *SIAM Journal on Scientific Computing*, 37(5):A2424–A2450.

[Grasedyck et al., 2013] Grasedyck, L., Kressner, D., and Tobler, C. (2013). A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78.

[Grosso et al., 1996] Grosso, R., Ertl, T., and Aschoff, J. (1996). Efficient data structures for volume rendering of wavelet-compressed data. In *Proceedings Winter School of Computer Graphics*. Computer Society Press.

[Grout et al., 2011] Grout, R., Gruber, A., Yoo, C., and Chen, J. (2011). Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow. *Proceedings of the Combustion Institute*, 33(1):1629 – 1637.

[Guthe et al., 2002] Guthe, S., Wand, M., Gonser, J., and Strasser, W. (2002). Interactive rendering of large volume data sets. In *Proceedings IEEE Visualization*, pages 53–60. Computer Society Press.

[Hackbusch, 2012] Hackbusch, W. (2012). *Tensor spaces and numerical tensor calculus*, volume 42 of *Springer series in computational mathematics*. Springer, Heidelberg.

[Hadwiger et al., 2012a] Hadwiger, M., Beyer, J., Jeong, W.-K., and Pfister, H. (2012a). Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2285–2294.

[Hadwiger et al., 2012b] Hadwiger, M., Sicat, R., Beyer, J., Krüger, J., and Möller, T. (2012b). Sparse PDF maps for non-linear multi-resolution image operations. *ACM Transactions on Graphics*, 31(6):133:1–12.

[Handschuh, 2015] Handschuh, S. (2015). *Numerical methods in tensor networks*. Dissertation, Universität Leipzig, Leipzig.

[Harshman, 1970] Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi–modal factor analysis. *UCLA working papers in phonetics*, 16:1–84.

[He et al., 2010] He, K., Sun, J., and Tang, X. (2010). Guided image filtering. In *European Conference on Computer Vision: Part I*, pages 1–14.

[Heckbert, 1986] Heckbert, P. S. (1986). Filtering by repeated integration. In *Proceedings ACM SIGGRAPH*, pages 315–321.

[Heinrich and Weiskopf, 2013] Heinrich, J. and Weiskopf, D. (2013). State of the art of parallel coordinates. In *Proceedings Eurographics 2013 (State of the Art Reports)*, pages 95–116.

[Hensley et al., 2005] Hensley, J., Scheuermann, T., Coombe, G., Singh, M., and Lastra, A. (2005). Fast summed-area table generation and its applications. *Computer Graphics Forum*, 24:547–555.

[Herman and Usher, 2017] Herman, J. and Usher, W. (2017). SALib: An open-source python library for sensitivity analysis. *The Journal of Open Source Software*, 2(9).

[Hitchcock, 1927] Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6:164–169.

[Hoeffding, 1948] Hoeffding, W. (1948). A class of statistics with asymptotically normal distribution. *The Annals of Mathematical Statistics*, 19(3):293–325.

[Homma and Saltelli, 1996] Homma, T. and Saltelli, A. (1996). Importance measures in global sensitivity analysis of nonlinear models. *Reliability Engineering & System Safety*, 52(1):1–17.

[Hooker, 2004] Hooker, G. (2004). Discovering additive structure in black box functions. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 575–580.

[Ihm and Park, 1999] Ihm, I. and Park, S. (1999). Wavelet-based 3D compression scheme for interactive visualization of very large volume data. *Computer Graphics Forum*, 18(1):3–15.

[Iooss and Lemaître, 2015] Iooss, B. and Lemaître, P. (2015). *A Review on Global Sensitivity Analysis Methods*, pages 101–122. Springer US, Boston, MA.

[Jeong et al., 2009] Jeong, W., Beyer, J., Hadwiger, M., Vazquez, A., Pfister, H., and Whitaker, R. (2009). Scalable and interactive segmentation and visualization of neural processes in em datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1505–1514.

[Kass and Solomon, 2010] Kass, M. and Solomon, J. (2010). Smoothed local histogram filters. *ACM Transactions on Graphics*, 29(4):100:1–10.

[Kazeev et al., 2014] Kazeev, V., Khammash, M., Nip, M., and Schwab, C. (2014). Direct solution of the chemical master equation using quantized tensor trains. *PLoS Computational Biology*, 10(3):1–19.

[Kazeev and Oseledets, 2013] Kazeev, V. A. and Oseledets, I. V. (2013). The tensor structure of a class of adaptive algebraic wavelet transforms. Preprint 2013-28, ETH SAM, Zürich.

[Kehrer and Hauser, 2013] Kehrer, J. and Hauser, H. (2013). Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):495–513.

[Kenett and Zacks, 1998] Kenett, R. and Zacks, S. (1998). *Modern Industrial Statistics: Design and Control of Quality and Reliability*. Duxbury Press.

[Kolda and Bader, 2009] Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

[Konakli and Sudret, 2015] Konakli, K. and Sudret, B. (2015). Low-Rank Tensor Approximations for Reliability Analysis. In *International Conference on Applications of Statistics and Probability in Civil Engineering*.

[Konakli and Sudret, 2016] Konakli, K. and Sudret, B. (2016). Global sensitivity analysis using low-rank tensor approximations. *Reliability Engineering & System Safety*, 156:64 – 83.

[Kressner et al., 2013] Kressner, D., Steinlechner, M., and Vandereycken, B. (2013). Low-rank tensor completion by riemannian optimization. *BIT Numerical Mathematics*, pages 1–22.

[Kroonenberg and Leeuw, 1980] Kroonenberg, P. and Leeuw, J. (1980). Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45(1):69–97.

[Kroonenberg, 1983] Kroonenberg, P. M. (1983). *Three-mode Principal Component Analysis: Theory and Applications*. Leiden: DSWO Press.

[Lakshminarasimhan et al., 2011] Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., and Samatova, N. F. (2011). *Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-temporal Data*, volume 1, pages 366–379.

[Lamboni et al., 2013] Lamboni, M., Iooss, B., Popelin, A., and Gamboa, F. (2013). Derivative-based global sensitivity measures: General links with sobol indices and numerical tests. *Mathematics and Computers in Simulation*, 87:45–54.

[Lee and Cichocki, 2017] Lee, N. and Cichocki, A. (2017). Fundamental tensor operations for large-scale data analysis using tensor network formats. *Multidimensional Systems and Signal Processing*, pages 1–40.

[Lee and Shen, 2013] Lee, T.-Y. and Shen, H.-W. (2013). Efficient local statistical analysis via integral histograms with discrete wavelet transform. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2693–2702.

[Lindstrom, 2014] Lindstrom, P. (2014). Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization & Computer Graphics*, 20(12):2674–2683.

[Lippert et al., 1997] Lippert, L., Gross, M., and Kurmann, C. (1997). Compression domain volume rendering for distributed environments. In *Proceedings EUROGRAPHICS*, pages 95–108. also in Computer Graphics Forum 16(3).

[Litvinenko et al., 2013] Litvinenko, A., Matthies, H. G., and El-Moselhy, T. A. (2013). *Sampling and Low-Rank Tensor Approximation of the Response Surface*, pages 535–551. Springer Berlin Heidelberg.

[Lundstrom et al., 2006] Lundstrom, C., Ljung, P., and Ynnerman, A. (2006). Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1570–1579.

[Marrel et al., 2009] Marrel, A., Iooss, B., Laurent, B., and Roustant, O. (2009). Calculations of Sobol indices for the Gaussian process metamodel. *Reliability Engineering and System Safety*, 94:742–751.

[McKay et al., 1979] McKay, M. D., Beckman, R. J., and Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61.

[Mikhalev and Oseledets, 2015] Mikhalev, A. Y. and Oseledets, I. V. (2015). Rectangular maximum-volume submatrices and their applications. arXiv preprint 1502.07838.

[Narita et al., 2011] Narita, A., Hayashi, K., Tomioka, R., and Kashima, H. (2011). Tensor Factorization Using Auxiliary Information. In Gunopulos, D., Hofmann, T., Malerba, D., and Vazirgiannis, M., editors, *Machine Learning and Knowledge Discovery in Databases*, volume 25, pages 501–516. Springer Berlin Heidelberg.

[Nehab et al., 2011] Nehab, D., Maximo, A., Lima, R. S., and Hoppe, H. (2011). GPU-efficient recursive filtering and summed-area tables. *ACM Transactions on Graphics*, 30(6):176:1–11.

[Nocke et al., 2007] Nocke, T., Flechsig, M., and Böhm, U. (2007). Visual exploration and evaluation of climate-related simulation data. In *Simulation Conference*, pages 703–711.

[Novikov et al., 2015] Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. (2015). Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 442–450.

[Novikov et al., 2014] Novikov, A., Rodomanov, A., Osokin, A., and Vetrov, D. (2014). Putting MRFs on a tensor train. In Jebara, T. and Xing, E. P., editors, *Proceedings 31st International Conference on Machine Learning (ICML)*, pages 811–819.

[Novikov et al., 2016] Novikov, A., Trofimov, M., and Oseledets, I. (2016). Exponential machines. *arXiv preprint*.

[Nugteren and Codreanu, 2015] Nugteren, C. and Codreanu, V. (2015). Cltune: A generic auto-tuner for opencl kernels. In *International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 195–202.

[Oferkin et al., 2015] Oferkin, I., Zheltkov, D., Tyrtyshnikov, E., Sulimov, A., Kutov, D., and Sulimov, V. (2015). Evaluation of the docking algorithm based on tensor train global optimization. *Bulletin of the South Ural State University: Mathematical Modelling and Programming*, 8(4):83–99.

[Oseledets and Tyrtyshnikov, 2011] Oseledets, I. and Tyrtyshnikov, E. (2011). Algebraic wavelet transform via quantics tensor train decomposition. *SIAM Journal on Scientific Computing*, 33(3):1315–1328.

[Oseledets, 2011] Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

[Oseledets et al., 2008] Oseledets, I. V., Savostianov, D. V., and Tyrtyshnikov, E. E. (2008). Tucker dimensionality reduction of three-dimensional arrays in linear time. *SIAM Journal on Matrix Analysis and Applications*, 30(3):939–956.

[Oseledets and Tyrtyshnikov, 2010a] Oseledets, I. V. and Tyrtyshnikov, E. E. (2010a). Tensor-tree decomposition does not need a tree. In *Linear Algebra and its Applications*.

[Oseledets and Tyrtyshnikov, 2010b] Oseledets, I. V. and Tyrtyshnikov, E. E. (2010b). TT-cross approximation for multidimensional arrays. *Linear Algebra Applications*, 432(1):70–88.

[Owen et al., 2013] Owen, A., Dick, J., and Chen, S. (2013). Higher order Sobol indices. *ArXiv e-prints*.

[Owen, 2013] Owen, A. B. (2013). Variance components and generalized Sobol indices. *SIAM Journal on Uncertainty Quantification*, 1(1):19–41.

[Owen, 2014] Owen, A. B. (2014). Sobol indices and shapley value. *SIAM/ASA Journal on Uncertainty Quantification*, 2(1):245–251.

[Porikli, 2005] Porikli, F. (2005). Integral histogram: A fast way to extract histograms in cartesian spaces. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 829–836.

[Potter et al., 2009] Potter, K., Wilson, A., Bremer, P.-T., Williams, D., Doutriaux, C., Pascucci, V., and Johnson, C. R. (2009). Ensemble-vis: A framework for the statistical visualization of ensemble data. In *IEEE International Conference on Data Mining Workshops.*, pages 233–240.

[Pretorius et al., 2015] Pretorius, A., Zhou, Y., and Ruddle, R. (2015). Visual parameter optimisation for biomedical image processing. *BioMed Central Bioinformatics*, 16(11):1–13.

[Queipo et al., 2005] Queipo, N. V., Haftka, R. T., Shyy, W., Goel, T., Vaidyanathan, R., and Tucher, P. K. (2005). Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41:1–28.

[Rai, 2014] Rai, P. (2014). *Sparse Low Rank Approximation of Multivariate Functions – Applications in Uncertainty Quantification*. Doctoral thesis, Ecole Centrale Nantes.

[Rajwade et al., 2013] Rajwade, A., Rangarajan, A., and Banerjee, A. (2013). Image denoising using the higher order singular value decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(4):849–862.

[Rakhuba and Oseledets, 2015] Rakhuba, M. V. and Oseledets, I. V. (2015). Fast multidimensional convolution in low-rank tensor formats via cross approximation. *SIAM J. Scientific Computing*, 37(2).

[Rodler, 1999] Rodler, F. (1999). Wavelet based 3D compression with fast random access for very large volume data. In *Proceedings Pacific Graphics*, pages 108–117.

[Ruiters and Klein, 2009] Ruiters, R. and Klein, R. (2009). BTF compression via sparse tensor decomposition. *Computer Graphics Forum*, 28(4):1181–1188.

[Saltelli et al., 2008] Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S. (2008). *Global Sensitivity Analysis: The Primer*. John Wiley & Sons, Ltd.

[Saltelli et al., 2004] Saltelli, A., Tarantola, S., Campolongo, F., and Ratto, M. (2004). *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Halsted Press, New York, NY, USA.

[Savostyanov and Oseledets, 2011] Savostyanov, D. V. and Oseledets, I. V. (2011). Fast adaptive interpolation of multi-dimensional arrays in tensor train format. In *Proceedings 7th International Workshop on Multidimensional Systems (nDS)*.

[Schlegel et al., 2011] Schlegel, P., Makhinya, M., and Pajarola, R. (2011). Extinction-based shading and illumination in GPU volume ray-casting. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1795–1802.

[Schneider and Westermann, 2003] Schneider, J. and Westermann, R. (2003). Compression domain volume rendering. In *Proceedings IEEE Visualization*, pages 293–300.

[Sicat et al., 2014] Sicat, R., Krüger, J., Möller, T., and Hadwiger, M. (2014). Sparse PDF volumes for consistent multi-resolution volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2417–2426.

[Silva and Herrmann, 2015] Silva, C. D. and Herrmann, F. J. (2015). Optimization on the hierarchical Tucker manifold - applications to tensor completion. *Linear Algebra and its Applications*, 481:131–173.

[Sobol, 2003] Sobol, I. (2003). Theorems and examples on high dimensional model representation. *Reliability Engineering & System Safety*, 79(2):187 – 193.

[Sobol, 1990] Sobol, I. M. (1990). Sensitivity estimates for nonlinear mathematical models (in Russian). *Mathematical Models*, 2:112–118.

[Soize and Ghanem, 2004] Soize, C. and Ghanem, R. (2004). Physical systems with random uncertainties: Chaos representations with arbitrary probability measure. *SIAM Journal on Scientific Computing*, 26(2):395–410.

[Soltészová et al., 2017] Soltészová, V., Birkeland, Å., Stoppel, S., Viola, I., and Bruckner, S. (2017). Output-sensitive filtering of streaming volume data. *Computer Graphics Forum*, 36(1):249–262.

[Sorber et al.,] Sorber, L., Van Barel, M., and De Lathauwer, L. Tensorlab v1.0. http://esat.kuleuven.be/sista/tensorlab/.

[Steinlechner, 2015] Steinlechner, M. (2015). Riemannian optimization for high-dimensional tensor completion. Technical Report 05.2015, Mathematics Institute of Computational Science and Engineering, EPFL.

[Sudret, 2008] Sudret, B. (2008). Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering and System Safety*, 93(7):964 – 979.

[Suter et al., 2013] Suter, S., Makhynia, M., and Pajarola, R. (2013). TAMRESH: Tensor approximation multiresolution hierarchy for interactive volume visualization. *Computer Graphics Forum*, 32(3pt2):151–160.

[Suter et al., 2011a] Suter, S. K., Iglesias Guitián, J. A., Marton, F., Agus, M., Elsener, A., Zollikofer, C. P., Gopi, M., Gobbetti, E., and Pajarola, R. (2011a). Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143.

[Suter et al., 2011b] Suter, S. K., Iglesias Guitián, J. A., Marton, F., Agus, M., Elsener, A., Zollikofer, C. P., Gopi, M., Gobbetti, E., and Pajarola, R. (2011b). Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143.

[Suter et al., 2010a] Suter, S. K., Zollikofer, C. P., and Pajarola, R. (2010a). Application of tensor approximation to multiscale volume feature representations. In *Proceedings Vision, Modeling and Visualization*, pages 203–210.

[Suter et al., 2010b] Suter, S. K., Zollikofer, C. P., and Pajarola, R. (2010b). Multiscale tensor approximation for volume data. Technical Report IFI-2010.04, Department of Informatics, University of Zürich.

[Tapia, 2011] Tapia, E. (2011). A note on the computation of high-dimensional integral images. *Pattern Recognition Letters*, 32(2):197–201.

[Tatu et al., 2012] Tatu, A., Maaß, F., Färber, I., Bertini, E., Schreck, T., Seidl, T., and Keim, D. A. (2012). Subspace search and visualization to make sense of alternative clusterings in high-dimensional data. In *Procedings IEEE Symposium on Visual Analytics Science and Technology*, pages 63–72.

[Ten Berge et al., 1987] Ten Berge, J. M. F., De Leeuw, J., and Kroonenberg, P. M. (1987). Some additional results on principal components analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 52:183–191.

[Treib et al., 2012] Treib, M., Bürger, K., Reichl, F., Meneveau, C., Szalay, A., and Westermann, R. (2012). Turbulence visualization at the terascale on desktop PCs. *IEEE Transactions on Visualization and Computer Graphics (Proc. Scientific Visualization 2012)*, 18(12):2169–2177.

[Tsai, 2009] Tsai, Y.-T. (2009). *Parametric representations and tensor approximation algorithms for real-time data-driven rendering*. PhD thesis, National Chiao Tung University.

[Tsai, 2015] Tsai, Y.-T. (2015). Multiway K-clustered tensor approximation: Toward high-performance photorealistic data-driven rendering. *ACM Transactions on Graphics*, 34(5):157:1–15.

[Tsai and Shih, 2012] Tsai, Y.-T. and Shih, Z.-C. (2012). K-clustered tensor approximation: A sparse multilinear model for real-time rendering. *ACM Transactions on Graphics*, 31(3).

[Tucker, 1966] Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311.

[Tyrtyshnikov, 2000] Tyrtyshnikov, E. E. (2000). Incomplete cross approximation in the mosaic–skeleton method. *Computing*, 64(4):367–380.

[Vasilescu and Terzopoulos, 2007] Vasilescu, M. A. O. and Terzopoulos, D. (2007). Multilinear projection for appearance-based recognition in the tensor framework. In *IEEE International Conference on Computer Vision*, pages 1–8.

[Vervliet et al., 2014] Vervliet, N., Debals, O., Sorber, L., and Lathauwer, L. D. (2014). Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis. *IEEE Signal Processing Magazine*, 31(5):71–79.

[Vlasic et al., 2005] Vlasic, D., Brand, M., Pfister, H., and Popović, J. (2005). Face transfer with multilinear models. In *ACM SIGGRAPH*, pages 426–433, New York, NY, USA. ACM.

[Wang and Ahuja, 2004] Wang, H. and Ahuja, N. (2004). Compact representation of multidimensional data using tensor rank-one decomposition. In *Proceedings International Conference on Pattern Recognition*, pages 44–47.

[Wei and Tao, 2010] Wei, Y. and Tao, L. (2010). Efficient histogram-based sliding window. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 3003–3010.

[Weiss, 2006] Weiss, B. (2006). Fast median and bilateral filtering. *ACM Transactions on Graphics*, 25(3):519–526.

[Wetzstein et al., 2012] Wetzstein, G., Lanman, D., Hirsch, M., and Raskar, R. (2012). Tensor displays: Compressive light field synthesis using multilayer displays with directional backlighting. *ACM Transactions on Graphics*, 31(4):80:1–11.

[Wu et al., 2008] Wu, Q., Xia, T., Chen, C., Lin, H.-Y. S., Wang, H., and Yu, Y. (2008). Hierarchical tensor approximation of multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):186–199.

[Wu et al., 2007] Wu, Q., Xia, T., and Yu, Y. (2007). Hierarchical tensor approximation of multidimensional images. In *Proceedings IEEE International Conference in Image Processing*, volume 4, pages IV–49–IV–52.

[Wu et al., 2000] Wu, Y.-L., Agrawal, D., and El Abbadi, A. (2000). Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes. In *Proceedings International Conference on Information and Knowledge Management*, pages 414–421.

[Xu et al., 2010] Xu, L., Lee, T.-Y., and Shen, H.-W. (2010). An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216–1224.

[Yeo and Liu, 1995] Yeo, B.-L. and Liu, B. (1995). Volume rendering of DCT-based compressed 3D scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43.

[Zhang et al., 2015a] Zhang, X., Xu, Z., Jia, N., Yang, W., Feng, Q., Chen, W., and Feng, Y. (2015a). Denoising of 3D magnetic resonance images by using higher-order singular value decomposition. *Medical Image Analysis*, 19(1):75–86.

[Zhang et al., 2015b] Zhang, Z., Yang, X., Oseledets, I. V., Karniadakis, G. E., and Daniel, L. (2015b). Enabling high-dimensional hierarchical uncertainty quantification by ANOVA and tensor-train decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(1):63–76.

[Zhao et al., 2016] Zhao, Q., Zhou, G., Xie, S., Zhang, L., and Cichocki, A. (2016). Tensor ring decomposition. *CoRR*, abs/1606.05535.

CURRICULUM VITAE

Personal Information

Name	Rafael Ballester-Ripoll
Date of birth	July 31st, 1989
Place of birth	Gandia, Spain

Education

2012-2017: Doctoral studies in Informatics at the Department of Informatics (University of Zurich), Zurich, Switzerland

2006-2012: Dual degree (BSc and MSc Mathematics; BSc and MSc Computer Science) at Universitat Politècnica de Catalunya, Barcelona, Spain

2002-2006: Secondary education at I. E. S. Antoni Llidó, Xàbia, Spain

Publications

Journal Articles

R. Ballester-Ripoll, E. G. Paredes, R. Pajarola. Sobol Tensor Trains for Global Sensitivity Analysis [Under review in *Reliability Engineering and System Safety*].

R. Ballester-Ripoll, E. G. Paredes, R. Pajarola. Tensor Algorithms for Advanced Sensitivity Metrics [Accepted in *SIAM Journal on Uncertainty Quantification*].

R. Ballester-Ripoll, D. Steiner, R. Pajarola. Multiresolution Volume Filtering in the Tensor Compressed Domain. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

R. Ballester-Ripoll, R. Pajarola. Tensor Decompositions for Integral Histogram Compression and Look-Up. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

R. Ballester-Ripoll, R. Pajarola. Lossy Volume Compression Using Tucker Truncation and Thresholding. *The Visual Computer*; 31: 1-14, 2015.

R. Ballester-Ripoll, S. K. Suter, R. Pajarola. Analysis of Tensor Approximation for Compression-Domain Volume Visualization. *Computers & Graphics*; 47: 34-47, 2015.

Conference Publications

R. Ballester-Ripoll, E. G. Paredes, R. Pajarola. A Surrogate Visualization Model Using the Tensor Train Format. *SIGGRAPH Asia Symposium on Visualization* (Macao, China); Dec. 2016.

R. Ballester-Ripoll, R. Pajarola. Compression of BTFs Using the Tensor Train Decomposition. *Pacific Graphics Short Papers* (Okinawa, Japan); Oct. 2016.

Miscellaneous

B. Flueckiger, N. Evirgen, E. G. Paredes, R. Ballester-Ripoll, R. Pajarola. Deep Learning Tools for Foreground-Aware Analysis of Film Colors [abstract]. *Workshop on Computer Vision in Digital Humanities* (Montreal, Canada), Aug. 2017.

R. Ballester-Ripoll, R. Pajarola. Tensor Methods in Visual Computing [tutorial]. *IEEE Visualization* (Baltimore, USA), Oct. 2016.

R. Ballester-Ripoll, S. K. Suter, R. Pajarola. Analysis of Tensor Approximation for Compression-Domain Volume Visualization [talk]. *Spring Conference on Computer Graphics* (Smolenice, Slovakia), Apr. 2016.

R. Ballester-Ripoll. Visual Data Processing in the Tensor Compressed Domain [talk]. *Workshop on Tensor Decompositions and Applications* (Leuven, Belgium), Jan. 2016.